Date of acceptance Grade

Instructor

Security Framework for the Web of IoT Platforms

Atarah Ivan Akoribila

Helsinki September 30, 2017

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Atarah Ivan Akoribila

Työn nimi — Arbetets titel — Title
Security Framework for the Web of IoT Platforms

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
	September 30, 2017	61

Tiivistelmä — Referat — Abstract

Connected devices of IoT platforms are known to produce, process and exchange vast amounts of data, most of it sensitive or personal, that need to be protected. However, achieving minimal data protection requirements such as confidentiality, integrity, availability and non-repudiation in IoT platforms is a non-trivial issue. For one reason, the trillions of interacting devices provide larger attack surfaces. Secondly, high levels of personal and private data sharing in this ubiquitous and heterogeneous environment require more stringent protection. Additionally, whilst interoperability fuels innovation through cross-platform data flow, data ownership is a concern. This calls for categorizing data and providing different levels of access control to users known as global and local scopes. These issues present new and unique security considerations in IoT products and services that need to be addressed to enable wide adoption of the IoT paradigm.

This thesis presents a security and privacy framework for the Web of IoT platforms that addresses end-to-end security and privacy needs of the platforms. It categorizes platforms' resources into different levels of security requirements and provides appropriate access control mechanisms.

ACM Computing Classification System (CCS):
D.4.6[Distributed systems]: Security and Protection-Access Controls

Avainsanat — Nyckelord — Keywords
Internet of things, security and privacy, heterogeneous networks

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

1.1 Background

The Internet of Things (IoT) paradigm aims at interconnecting smart and autonomous devices using the Internet and other heterogeneous networks in order to share data and enable remote management [GTM11]. The basic idea underlying the paradigm is to tap into the pervasive presence of sensors, actuators and communication technologies to uniquely identify objects and interconnect them in such a way that will enable cooperation towards achieving common goals [AIM10]. It envisions a future of smart spaces such as smart homes and smart cities wherein seamless interactions between sensors and actuators will provide self-reliant services for end-users. The paradigm has received great interest in the past years and research predict that IoT devices connected to the Internet will triple from the current estimate of 10 billion to 34 billion by 2020 [Meo16]. Collectively, these connected devices will be producing and consuming petabytes of data in near real-time.

The early approach to realizing the paradigm's objectives was by use of middleware solutions called *IoT platforms* to provide services such as networking, messaging and security needed to connect the *smart objects* in the physical while providing APIs for application development. Examples of these IoT platforms include solutions such as *Xively*[1], *Arrayent Connect*[2] *and ThingWorx*[3].

Though the middleware solutions helped in bringing IoT goals closer, some challenges emerged with their introduction. Prominent among these challenges is lack of existing standards on how IoT platforms expose services that could be consumed by devices external to them or even other platforms. Consequently, the middleware efforts produced closed and vertical systems with little interactions where one IoT platform produces its own hardware and lock them in with their proprietary middleware and user applications. A result of this is a plethora of islands of non-interoperable IoT platforms, each competing for dominance [BL14]. Moreover, application development is hindered since developers have to deal with interoperability problems themselves in order to create innovative apps. This impedes innovation in the IoT ecosystem and its wide adoption. To address these issues, efforts are being made to interconnect the different IoT platforms into a Web of heterogeneous

[1] `https://www.xively.com`
[2] `http://www.arrayent.com/platform/`
[3] `https://www.thingworx.com`

IoT hubs called *Web of IoT platforms* [MMST16]. For example, Blackstock and Lea [BL14] have proposed to use a catalogue for querying and representing IoT resources (URIs) on the Web resulting in the Hypercat interoperability specification[4].

This specification moves IoT ecosystem towards the desired interoperability by requiring hubs to expose their resources using Web architecture and RESTful Web services (Web of things). In the next stage, exposed resources from hubs are required to meet agreed specifications which results in the ability to develop adapters and integration tools for hubs to inter-operate. The Hypercat's approach further enables a catalogued representation of hub's resources that can be queried by applications from different hubs thereby enabling seamless interoperability.

Another proposal in [MT15] uses the concept of IoT hubs to extend existing middleware solutions with the IoT Hub REST API framework that enable sharing of resources. The REST API can be implemented on any existing IoT platform/middleware to support interoperability. Furthermore, the work uses meta-hubs that expose resources provided by IoT Hubs in a similar fashion as in HyperCat.

In all these efforts, the goal is to tap into the ubiquitous nature of the Web, make IoT platforms produce services that conform to common Web standards thereby leading to seamless interoperability.

1.2 Motivation

Making it possible for different IoT platforms and hubs to interoperate through Web technologies is a great step towards the IoT ecosystem's goals. However, interoperability is only a part of a bigger problem hindering full take-off of IoT paradigm. One core research challenge is how to effectively manage the petabytes of data flows that will result from interactions between IoT devices, in terms of collection, storage, analytics and security [MWC13]. The most prominent among these is the issue of securing platforms and their data and at the same time exposing necessary services for interactions in the ecosystem. Even in the conventional World Wide Web where end-devices such as personal computers and servers are unarguably well resourced in terms of computational power and storage than most IoT devices, there has still been frequent hacking and malware attacks. Few examples of these attacks include the *HeartBleed*[5] bug that affected several giant IT companies, the 2014 hack on

[4] `http://www.hypercat.io`

[5] `http://heartbleed.com`

Yahoo that affected up to five million user accounts and the *WannaCry* ransomware that hit most parts of the world in May 2017 leading to disruptions in UK hospitals and other institutions in different counties.

We envision that when IoT fully takes off, highly sensitive user data such as flows from patient life support devices, autonomous vehicles, security services and many more will be part of the ecosystem. All these flows are also going to be at the mercy of cyberattacks. This calls for taking strong measures in handling security requirements in order to build a safe and trusted Web of IoT platforms. It is worthy pointing out that issues of security provision in IoT are more challenging than the ordinary Internet or Web. One reason for this is that every single device and sensor in the IoT represents a potential attack point of the ecosystem and since there are trillions of such devices and sensors participating, the result is an exponentially larger attack surface.

It can be deduced from the above examples of cyber attacks, coupled with the highly sensitive data resulting from IoT interactions, that security requirements have to be addressed adequately in the Web of IoT platforms. This is a basic requirement to gain user trust and grow the IoT ecosystem. Several studies have been conducted on IoT platforms. For instance Mineraud et al. [MMST16] provides a comprehensive survey and gap analysis in these platforms, however, to the best of our knowledge there is no comprehensive solution that addresses the cross-cutting security concerns in these platforms. Guaranteeing security in all aspects of interactions in IoT platforms is a prerequisite to interoperability. This therefore motivates the need for creating a security framework for the Web of IoT platforms already started in [MT15]. At the very minimum, the IoT architecture needs measures to handle security and privacy requirements such ***Provision of Authentication, Access control, Privacy and confidentiality*** [Web10]. We also envision that the Web of IoT platforms will embrace doing analytics and taking decisions at the edge of the networks hence the need to provide additional security requirement in the form of ***Securing offloaded computations (edge computing)***. However, it should be noted that the *Privacy and Confidentiality* requirement is left out of this thesis and proposed to be covered in future works.

The main contribution of this thesis is to provision a comprehensive security framework for the Web of IoT platforms. The framework lays out how existing security techniques could be harnessed to provide the above mentioned security and privacy needs. This will secure interactions between different IoT platforms such as ex-

changing data or offloading computational tasks, while making sure that producers remain in full control their data.

The reminder of this thesis is as follows: Section 2 gives an overview of IoT platforms and their roles. It also discusses the Web of IoT platforms and how to extend IoT platforms for distributed analytics. Section 3 presents state-of-the-art security techniques that can be leveraged for the security framework. Section 4 describes the security framework and finally discussions and conclusions are presented in sections 5 and 6 respectively.

2 IoT platforms

Essentially, IoT platforms are middleware and infrastructure that provide interconnections between smart objects and enable interactions between these smart objects and end-users [MMST16]. The platforms serve as the glue between the various hardware and software components of the IoT ecosystem, providing several support functions such as *Connection, Communication* and *Data Management*.

The rest of the section discusses some common types of IoT platforms in use and the different patterns of interactions within these platforms which make up the Web of IoT platforms.

2.1 Types of IoT platforms

Platforms can be put into two broad categories based on their mode of deployment. Firstly, we have *local platforms* whose solutions are usually available within a local network and maintenance is done by the system admin of the network. They provide connectivity and server solutions to smart objects within the network and might not necessarily need Internet services. For example, with the aid of micro-controllers like Arduino boards, one can set up a local IoT platform that controls devices within a local setting without getting onto the Internet.

A second broad category of platforms is the *cloud-based (Software/Platform-as-a-Service)* model [MMST16]. Solutions in this category provide cloud-based computing services such as data storage, analytics and visualizations or tools that react to data. Platform-as-a-Service (PaaS) solutions provide integrated cloud-based platforms with pre-installed operating systems and application frameworks that take

care of all necessary infrastructure from hardware up to tools required to build end-user applications [ZCB10]. Examples include kaa[6], Arrayent Connect[7] and Axeda[8].

A second flavor of the cloud-based solution is *Software-as-a-Service (SaaS)* where end-user applications are provided as an extension to services provided by PaaS. For instance, the Nest platform[9] has mobile end-user applications to control home thermostats and cameras.

2.2 Interactions within the Web of IoT Platforms

In the Web of IoT platforms, there are different patterns of interactions within a single platform and between the different platforms which need to be understood in order to provide the needed security. We therefore describe these interactions in detail here and also bring up the security and privacy needs in each of them. We do this using the common patterns of interactions between IoT devices as specified in the guiding architectural document for networking smart objects (RFC 7452) [TM15]. It should be noted that some of these interactions such as *Device-to-Device, Platform-to-Platform* and *Edge Computing Interactions* are not in common use today. However, we envision them playing an important role in the implementation of the Web of IoT platforms hence the need to secure them.

2.2.1 Device-to-Device Pattern

Devices within this model establish connection to each other and directly communicate without the help of an intermediary application server (see Figure 1). The devices usually adhere to common protocols such as Bluetooth, ZigBee[10] or Z-Wave[11] that make it possible for them to directly understand exchanged data.

The model is common with scenarios where the interacting devices are within very close proximity and requires low data rate exchanges. We also envision that this pattern will increasingly become popular as the IoT paradigm takes off fully. Typical applications include fitness equipments such as heart rate monitors and cadence sensors or home automation systems. This model however, is a major source of

[6] `http://www.kaaproject.org`
[7] `http://www.arrayent.com/platform/`
[8] `http://www.ptc.com/axeda`
[9] `https://nest.com`
[10] `http://www.zigbee.org`
[11] `http://z-wavealliance.org`

Figure 1: Device-To-Device Communication pattern [TM15]

interoperability issues [KR15] as it usually leads to cases where device manufacturers implement specific data formats instead of open standard data formats. For example, the family of devices using ZigBee protocols are not natively compatible with those using Z-Wave. This non interoperability in the underlying device-to-device communication protocols limits a user's choice to devices within a particular protocol family.

In this pattern, the security needs will include how to effectively provide mutual authentication between devices directly communicating with each other without a bridge. There is also the need to have access control on a device's resources to prevent unintended exposure.

2.2.2 Device-to-Gateway Pattern

In this interaction pattern, the IoT devices use an application layer gateway as conduit to connect to a cloud service. The gateways run application software locally that provide security, protocol, or data translation. Data flows from the edge devices to the gateways using standard protocols such as CoAP, TLS, DTLS or Bluetooth. We also consider users[12] connecting to gateways/platforms to fall under this pattern. The gateway in turn transmits it to the cloud mostly in the form of Web services (see Figure 2). These gateways could also serve as platforms or dedicated "hubs" that are implemented with micro controllers such as Arduino boards[13] and Raspberry Pi[14]. In some consumer settings, smart phones with running apps serve as gateways

[12]Users can be human or machines
[13]`https://www.arduino.cc`
[14]`https://www.raspberrypi.org`

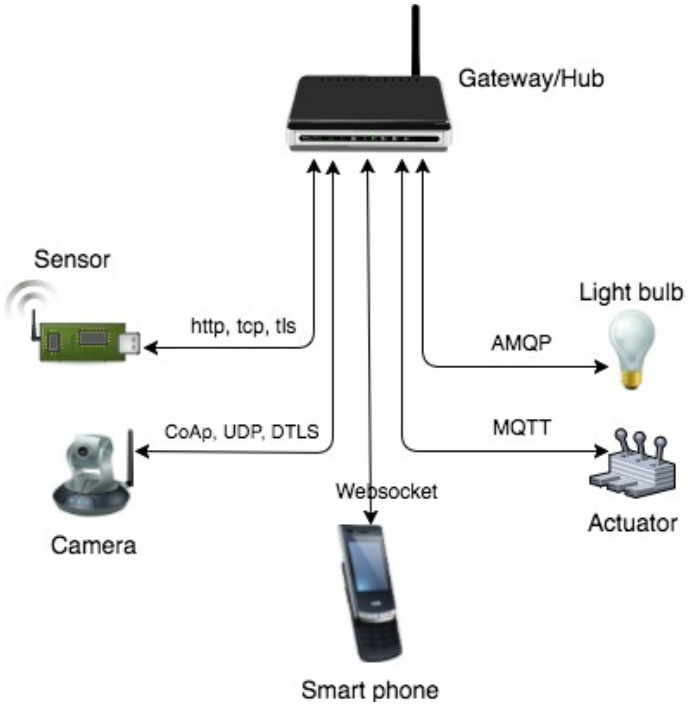

Figure 2: Device-to-Gateway pattern [TM15]

relaying data from edge devices to the cloud [KR15]. For example, in the Fitbit[15] ecosystem, Fitbit tracker syncs with a Fitbit app installed on a smart phone via Bluetooth. The tracker sends data to this app which in turn transmits it to the cloud service.

Regardless of which form of gateway is used, two sets of flows of data need protection namely from edge device to a local hub and from the hub to a cloud service. We need to provide security for these flows in terms of who is authenticated to have access to them and what kind of access rights to interact with them.

2.2.3 Device-to-Cloud Pattern

In this communication model, IoT devices connect directly to an Internet cloud service provider.

As shown in Figure 3, data and control message traffic exchanged between the devices and IP network will usually use existing technologies like Ethernet and

[15] `https://www.fitbit.com/fi`

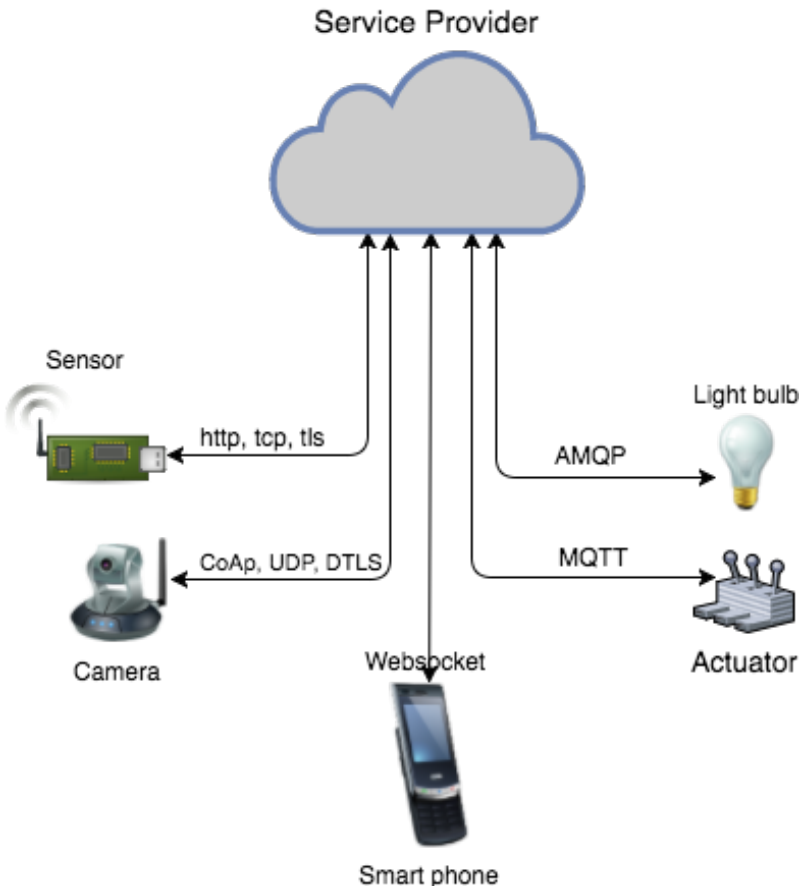

Figure 3: Device-to-Cloud pattern [TM15]

Wi-Fi. Consumer IoT providers like Nest Labs[16] use this model. One important use-case of this pattern is that, it enables constrained devices to seamlessly migrate computational tasks to more powerful servers in the cloud (Computational Offloading) [Est10].

Some security issues in this pattern include how to mutually authenticate devices and the cloud hosted platforms. We also need to be able to provide access controls to guard resources both on the cloud and device ends. Another consideration is how to protect the executions that constrained devices offload to cloud such as provision of safe execution environments so that nothing is leaked to unintended parties. The offloaded executions should also be prevented from accessing unintended resources while in the cloud platforms.

[16] `https://nest.com`

2.2.4 Platform-to-Platform Pattern

This enables back-end data sharing between different service providers. The pattern has not been fully developed and adopted by the different platform providers leading to the current vertical silos of IoT platforms. However, it is a very crucial part in implementing the Web of Things platforms. It forms the basis for interoperability and mash-ups in the IoT ecosystem and also serves as the means by which platforms get services from other platforms. For instance, through computational offloading, nodes in a given platform can seamlessly migrate a computational task to other nodes or more powerful servers in different platforms for execution. This is an important enabler in the IoT ecosystem since, by doing so, constraint devices are alleviated of restrictions in terms of their computational power, battery life and memory [PHG16]. Unlike the vertical silos created by Device-to-Cloud pattern,

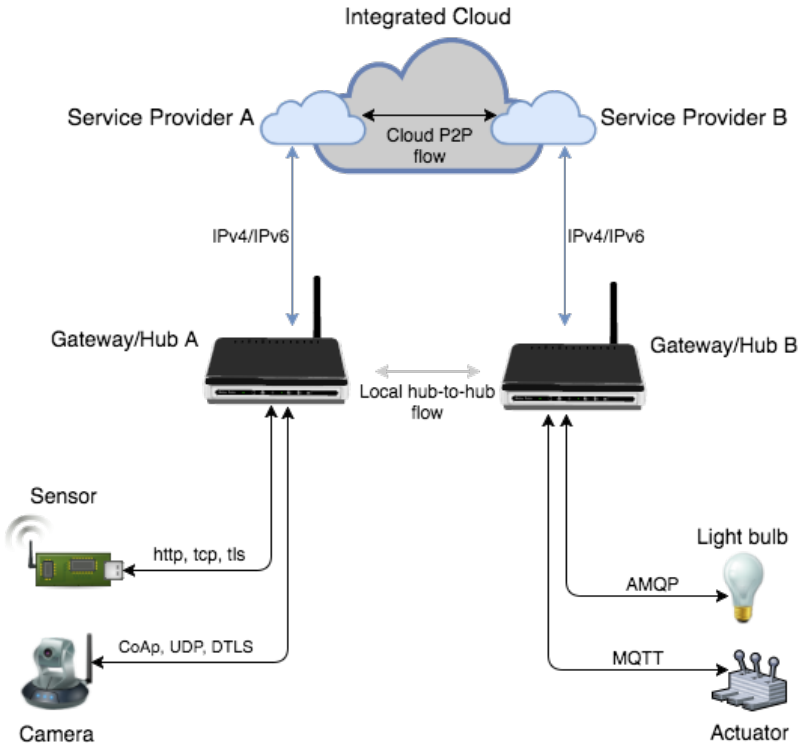

Figure 4: Platform-to-Platform pattern

back-end data sharing enables horizontal flow of data between different local hubs or cloud-based service providers (Figure 4).

To enable seamless interoperability between the different platforms, RESTful Web

APIs are used to expose relevant services. The primary flows in this pattern that needs protection are the "between-platform" exchanges of data, and this is usually done with federated authentication and authorization tools like OAuth 2.0. These protocols will be described in greater details in Section 3.

2.2.5 Edge Computing Interactions

Real-time analysis of data, especially data that comes in huge volumes from IoT devices, is one of the challenges introduced by the Internet of Things. The current trend of exporting all data into the cloud for analysis raises security and privacy concerns. For example, how do data owners remain in control of their data and ensure it is not exposed to third parties by the cloud owners without their consent? One way of mitigating security and privacy risks in exporting to the cloud is by making use of edge/fog computing. Through edge computing, IoT Hub platforms are able to cooperate and organize as micro-clouds to perform analytics very close to the data sources (edge of network). This does not only reduce the privacy and security risks in exporting to clouds but also reduces the network bandwidth needed in transporting the tremendous volumes of data to the cloud [AS16].

It should be noted that edge computing in IoT is not yet a full-fledged concept though platforms like Azure IoT Edge[17] have made some efforts towards sending analytics to edge devices. Nonetheless, we envision that edge computing will play an important role to improve performance in the Web of IoT platforms. In order to realize the benefits of edge computing in the Internet of Things, appropriate security measures must be put in place. For instance, foreign code to a given platform must be executed in secure environments, preventing it from interfering with platform's resources not exposed to it. There is also the need to ensure that a platform's code or data whilst seeking services in a foreign platform or micro-cloud is not tampered with. We will look at isolation measures that help secure edge interactions by providing secured environments to execute foreign code in Section 3.4.

[17] `https://azure.microsoft.com/en-us/campaigns/iot-edge/`

3 State-of-the-art Security Mechanisms of IoT Platforms

As discussed in Section 1.2, ensuring fundamental security requirements such as authentication, authorization, privacy and confidentially are fundamental to implementing the Web of IoT platforms. We also extend the requirements to securing edge interactions and remote execution of code in order to cover services such as edge computing and computational offloading for the Web. In this section, we discuss the security requirements listed above and bring up some existing protocols that can be leveraged to meet our security requirements. As part of the discussion on authorization, we consider how to secure remote execution of code resulting from computation offloading as described in Section 2.2.3.

3.1 Authentication

Authentication establishes identities for subjects in a system and verifies that the participating subjects in an interaction are authentic instances of the identities they claim to be or possess. It is the means by which clients and service providers prove to each other that they are acting on behalf of specific users or systems in a distributed environment [SBM+04]. Authentication methods have traditionally been based on one or more of the following properties:

1. What does the subject know? : In the case of passwords and pass-phrases authentication.
2. What is the subject or client? : Found in biometrics authentication.
3. What does the subject have? : Private keys and secret tokens authentication.

Authentication techniques can be categorised into different groups based on different metrics of interest. For instance, Saadeh et al. [SSQA16] have used metrics such as whether techniques are *centralized versus distributed or hierarchical versus flat.* In other instances, there may be the need for both parties taking part in a communication (client/server or peer-to-peer) to mutually authenticate each other. In this section, the authentication techniques are grouped and discussed based on the above factors and the needs of the Web of IoT platforms. Consequently we will have categories such as *mutual authentication, multi-factor, federated-based, certificate-based and framework-based* techniques. In each each of these categories, we discuss

state-of-the-art protocols and further look at other characteristics of interest such as whether it is a one-way or two-way (mutual authentication), central or distributed protocol. We also discuss the protocol's suitability to the IoT environment based on factors such as resource (computation power, memory, or energy) usage. Table 1 gives a summary comparison of the discussed protocols in terms of the main roles they play and their pros and cons.

Protocols	**Type**	**Main purpose**	**Pros**	**Cons**
Kerberos [NT94]	Federated	Identity verification of principals	1) Supports mutual authentication, 2) Eliminates transmission of un-encrypted passwords, 3) used as base for authorization protocols	1) Single point of failure without fallback servers 2) Restricted time requirement 3) Requires user accounts with authentication server
LDAP [Don02]	Mutual	Controls access to directory resources	1) Structured for hierarchical access to user information and easy to search based on all of this information. 2) Its extensions for other transport layer security protocols (SSL/TLS) makes sessions more secured	1) Not optimized for write operations 2) no standard specifications for access control on directory entries
SASL [IMG+07]	Framework	Provides authentication and data security protection for connection-oriented application protocols	1) Highly interoperable with many application level protocols 2) Flexibility in choice of mechanisms makes clients of varying capabilities participate 3) Security requirements of system can be tuned easily by changing mechanism 4) provides additional security layer service to protect data before transmission	1) Susceptible to mechanism downgrade attacks if no provisions are made to protect mechanism negotiating exchanges
Shibboleth [CS05]	Federated	Enables Single sign-on into federated systems	1) Allows access to remote resources using local credentials 2) Relieves users the burden of remembering different usernames and passwords 3) Avoids problems of proxy servers for remote access	1) Difficult for systems that want to avoid XML since it relies on SAML, 2) No global log-out
EAP [ABV+04]	Framework	Provides network access authentication	1) Different authentication mechanisms can be plugged in that constraint devices support 2) Flexibility allows for more secure authentication methods 3) Highly interoperable with different authentication protocols	1) EAP methods that permit plain text conversations are susceptible to attacks

Table 1: Comparison of Authentication Protocols

3.1.1 Certificate-based Authentication

These schemes are based on what a subject has (certificate) and so commonly use public key certificates to establish the authenticity of an entity seeking to be authenticated. To prevent masquerading a subject, the certificate is usually signed by a third party called Certificate Authority that vouches for its authenticity (CA).

To validate a certificate, the validating node must have have a certificate of the CA that issued the certificate. A certificate is an electronic document that gives a unique identity to an entity and associates that identity with a public key. Most certificates are organized according to X.509 v3 certificate specification [CSF+08] which partitions a certificate into data and signature sections. Data section includes information like X.509 version number supported, serial number of certificate, information on public key and associated algorithm, distinguished name (DN) of CA that issued it and certificate validity period. The signature section contains the digital signature of the CA which is obtained from hashing of the certificate's content and encrypting with the CA private key. The section also contains the corresponding cryptographic algorithm used by CA to create its digital signature.

One advantage of certificate-based schemes is that it enables usage of asymmetric keys which can be burned into secure storage on a device and never leaves the device hence more secured than usernames and passwords. It is therefore preferred in strict security requirements IoT applications when the participating devices can provision certificates.

Digital certificates are an important part of many security protocols. For instance, various versions of X.509 certificate specification are a basic part of protocols such as the Kerberos (see Figure 6) and SSL/TLS in their authenticating processes.

3.1.2 Mutual Authentication

In Mutual authentication schemes, the authenticating parties are required to prove their respective identities to each other before performing any application functions. Each party's identity can be proved through a trusted third party such as Certificate Authority (CA) or by use of cryptographic methods such as public key mechanisms. Examples of protocols used include IpSec, LDAP and so on.

***Secure Sockets Layer/Transport Layer Security (SSL/TLS)*:** This protocol sits between the application protocol layer and the TCP/IP layer. It secures

application data before it is passed onto the transport layer and supports several application layer protocols. It provides server authentication, data encryption and message integrity over TCP/IP connections [DR08]. Essentially, SSL/TLS authenticates participants to start communication and provides a secure pipe through which the communication takes place. The types of data encrypted include urls, http headers, cookies and form data.

SSL/TLS Protocol Layers: The SSL/TLS protocol essentially consists of two layers, the Handshake layer and the Record layer (see Figure 5). The Handshake layer is made up of three sub-protocols namely: Handshake Protocol, the Change Cipher Spec Protocol, and the Alert protocol [CDM03]. The Handshake sub-protocol gives a number of functions. It does a set of message exchanges that starts authentication and negotiates the encryption, hash, and compression algorithms. The protocol uses X.509 certificate 1. SSL/TLS uses public key encryption to authenticate the server to the client, and optionally the client to the server. Public key cryptography is also used to establish a session key. The session key is used in symmetric algorithms to encrypt the bulk of the data. This combines the benefit of asymmetric encryption for authentication with the faster, less processor-intensive symmetric key encryption for the bulk data.

The protocol at the record layer receives and encrypts data from the application layer and delivers it to the Transport Layer. The Record Protocol takes the data, fragments it to a size appropriate to the cryptographic algorithm, optionally compresses it (or, for data received, decompresses it), applies a MAC or HMAC (HMAC is supported only by TLS) and then encrypts (or decrypts) the data using the information negotiated during the Handshake Protocol.

***Lightweight Directory Access Protocol (LDAP)*:** LDAP is an application layer protocol that defines standard APIs to access and update a *directory* [TEG+06]. Its current specifications are defined in RFC 4511 [Ser06]. It supports both TCP and UDP at the transport layer. A *directory* is a special type of database containing entries of resources that are arranged in a tree-like structure called *Directory Information Tree (DIT)*. Each entry has a unique name called its *distinguished name (DN)* formed from relative distinguished names from parent nodes of that entry.

An LDAP client uses the *"bind", "unbind" and "abandon"* operations to establish sessions, terminate sessions and discard outstanding operations respectively with a server.

LDAP Security model is based on the bind operation together with the DN of

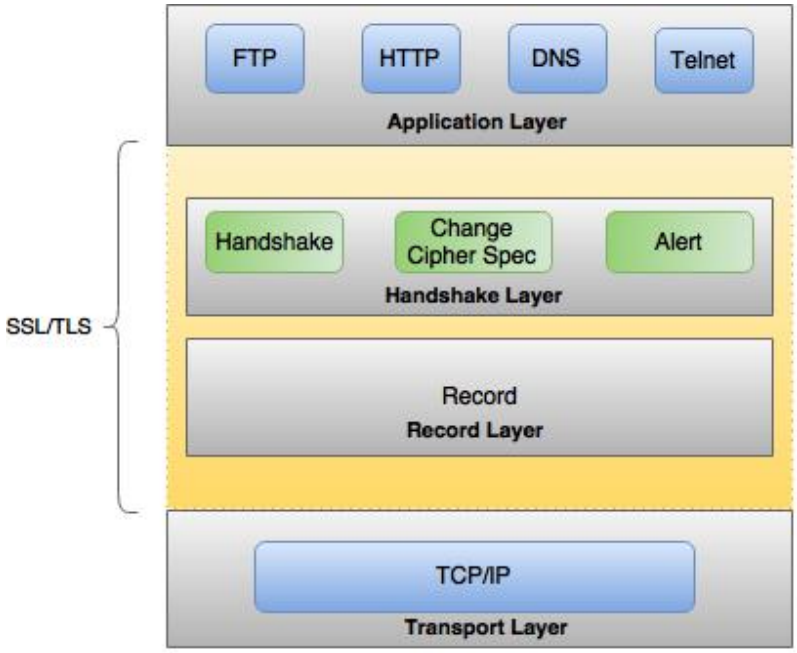

Figure 5: SSL/TLS Protocol Layers [DR08]

a directory. There are different bind operations that result in different levels of authentication on an LDAP server. The simplest form of authentication for a client is to supply its DN and a password to be checked for a matching entry in the directory. A non-authenticated and anonymous session is assumed by the server when no supplied DN or password. Another bind command supports the use of *SASL* mechanisms to support other stronger authentication methods like Kerberos. As an additional security mechanism, LDAP has extensions to use SSL/TLS at the transport layer to encrypt sessions [Ser06].

One notable lacking feature of the current specification of LDAP (version 3) is the lack of standards for setting access control on entries in an LDAP-enabled directory [Don02].

3.1.3 Multi-factor Authentication

Multi-factor authentication schemes seek to add an extra layer of protection between communicating parties by requiring the use of two or more of the factors of authentication described in the previous section. Most systems using this scheme

usually require users to first apply "what they know factor" in the form of username and password. Then a second step could be what the user possess (such as mobile phone with registered number) being used as second layer of protection. In this case, a code is sent to the user through the phone to be used for login. Examples of systems that use this scheme include *AWS Multi-Factor Authentication (MFA)*[18] and *Azure Multi-Factor Authentication*[19]. A good side of this is that a second layer of protection is added at little overhead in terms of higher computational power for cyrptographic functions. However, the scheme relies so much on the human intervention in the case of the use of the mobile phone and hence not very suitable where devices need to to authenticate each other with little human intervention.

3.1.4 Federated Identity Authentication

Federated Identity (FID) schemes allow delegation of authentication to an external identity provider. In FID, a user's credentials are hosted by an Identity provider or federation server different from the server hosting resources of interest. When the user wants access to a service, the service provider trusts the federation server to validate the user's credentials. In this case, a user's credentials are not directly provided to service provider or any other entity but the FID.

Examples of platforms that allow users to log onto third-party applications include *Microsoft account, Google Account, Yahoo and Twitter.* An advantage of this scheme is that it delegates all authentication tasks to bigger organizations with resources to handle. This makes it possible for resource constraint platforms in the IoT ecosystem to relieve themselves the burden of handling authentications. It also allows the separation of authentication from application code whilst allowing platforms to use wider range of identity providers (IdP). Federated schemes use protocols such as Kerberos, Open Authorization (OAuth) and Shibboleth.

***Kerberos*:** The Kerberos protocol is a distributed authentication service used to verify identities of principals [NT94], [NYHR05]. It is robust for different environments — including where there is no guarantee of protection of packets traversing the network and largely uses symmetric encryption, though some works [AJR11] have added extensions using Public-key cryptography.

In its basic operation, Kerberos service involves an Authentication Server (AP)

[18] `https://aws.amazon.com/iam/details/mfa/`

[19] `https://docs.microsoft.com/en-us/azure/multi-factor-authentication/multi-factor-authentication`

which includes a certificate granting component, an Application Server (AS) that holds desired resources and the client. A client sends a request to an AS for credentials to a given AP. The AS responds with encrypted credentials containing a ticket for the requested AP and a session key. Finally, the client sends a resource request together with the ticket to the AP and in the processes, the session key is shared. The session key is then used to authenticate the client (see Figure 6). The protocol also provides for authentication of a principal registered with a different authentication server than the verifier, know as *cross-realm authentication*.

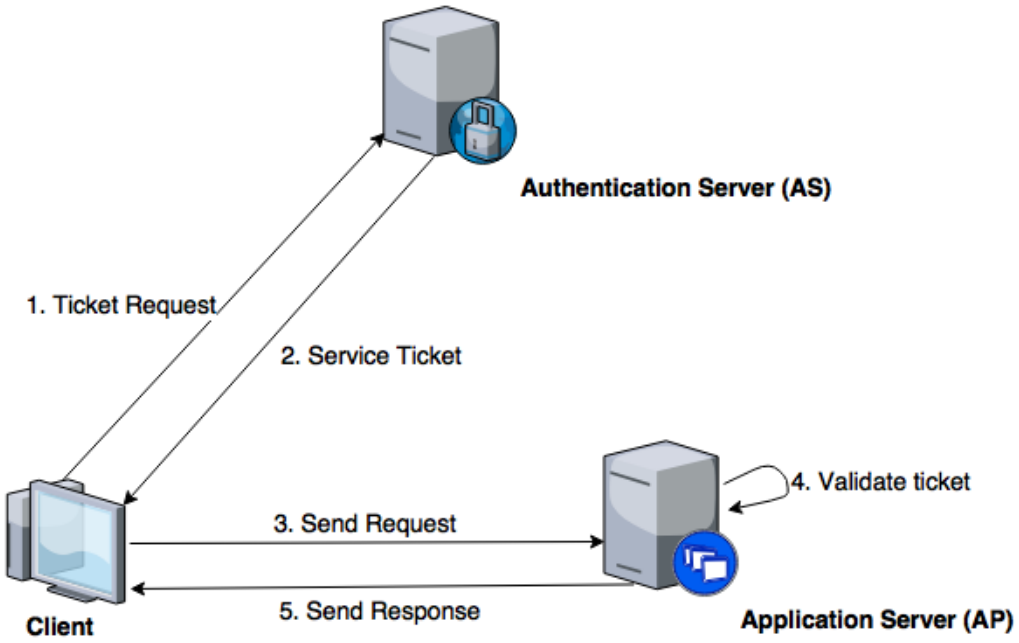

Figure 6: Kerberos authentication process [AJR11]

Kerberos can be used to pass authorization information from other services OAuth protocol and hence serves as a base for building authorization service. It also supports mutual authentication [AJR11] through the use of the session key — an enhanced security feature for high security-requirement scenarios. Another advantage of Kerberos is the elimination of transmission of unencrypted passwords across networks thereby reducing threats posed by packet sniffers.

Kerberos however also has its drawbacks as pointed out by works such as [LDJ14]. It requires user accounts with the authentication server to operate and expiration of the certificates also implies that the protocol has restricted time requirement. Since the protocol relies on continuous availability of a Kerberos authentication

server, a single point of failure is introduced unless multiple servers are introduced for fallback.

***OpenID Connect*:** OpenID Connect [SBJ+14] is an identity layer that enables single sign-on and provides a login session semantic on top of the OAuth protocol. It extends OAuth to enable End-Users to be Authenticated through an ID Token data structure represented as JSON Web Token (JWT) [JBS15]. The JWT represents an identity card that contains claims asserting the identity of the end-user, issuing authority, expiration and so on. It may optionally be encrypted for confidentiality.

Blazquez et al. [BTV15] looks at the performance evaluation of leveraging OpenId Connect in the IoT environment with the realization that though it represents an easy to use substitute, there is a performance dip in high-load environments. There is also the risk of man-in-the-middle attack since the same access token is re-used across different requests. This could be remedied through generation and verification of signatures at both ends per request. However, the remedy is at the risk of potential performance loss.

***Shibboleth*:** The Shibboleth protocol [CS05] is an open-source protocol specification that enables single sign-on into federated systems. It uses other standards such as SAML, PKI and HTTP. Its basic architecture is made up of the *user* who wants to use a protected resource, the *service provider (SP)* and an *identity provider (IdP).*

The process starts with a user requesting for a resource from the SP. With no active session, the SP redirects the user to an IdP with an issue Authentication Request. After successful identification of the user, the IdP makes an authentication response and sends it with the user back to the Service Provider. On a second arrival at the SP, validation of the response is made and a new session is created for the user to access the protected resource.

Shibboleth's reliance on the XML-based SAML [CS05] could be one of the cons for implementing systems that want to entirely avoid XML parsing. Some works also highlights an issue of "no-global" log-out where the protocol is able to assure log out from all the service at a time [KMRT15].

3.1.5 Framework-based Authentication

These schemes are frameworks that make it possible for different authentication protocols to be plugged-in and used. Most of these schemes permit run-time negotiation between the authenticating parties (client and server) on which specific protocol to

use. For example the SASL protocol framework described later in this section allows the SASL server to send the list of supported protocols to the client to select before authentication and subsequent exchanges start. This is an advantage for the IoT environment since devices vary in their ability to support different protocols. The scheme gives these devices the option to choose amongst a host of protocols that they can support. Extensible Authentication Protocol (EAP) is another example of this scheme mostly used in wireless networks and point-to-point connections.

Simple Authentication and Security Layer (SASL): SASL is a protocol framework that provides authentication and data security protection for connection-oriented protocols [MZ06]. The framework introduces an abstraction layer that decouples authentication mechanisms from the protocols thus making it possible for any supported authentication mechanisms to be plugged in and used by any protocol that implements SASL. This enables new protocols to seamlessly reuse existing mechanisms without redesign and vice versa. The protocol operates at the application layer, providing security for application protocols such as AMQP and XMPP. It is suitable for device-to-cloud, gateway-to-cloud or platform-to-platform flows, in which cases a platform's server will implement the protocol and host the authentication server for it.

SASL also optionally allows negotiation of security layer services to provide data integrity and confidentiality protection for subsequent exchanged data. A successful negotiation will have the server and client install the required security layer services that will be used to process protocol data into buffers of protected data before transmission.

SASL authentication process: In order to use SASL, each protocol exposes a method to identify a particular mechanism to be used, a method for which the chosen mechanism exchange server-challenges and client responses and a method for communicating the outcome of exchanges to the right parties.

Identity Concepts: SASL framework provides two identity concepts — *Authentication Identity*, which describes the authentication credentials of a client, and *Authorization Identity* that is used to act as a given subject. Authentication identity can take several forms such as X.509 certificates, Kerberos tickets or username/password. SASL also specifies which identity form to use for authorization identity, the syntax and semantics of the character string representing it.

A series of message exchanges take place to authenticate a client as shown in Figure 7.

1. A client sends request to an Authentication server (AS) to be authenticated together with its chosen mechanism from list advertised by the server.
2. The server responses with challenges depending on the chosen mechanism.
3. Client provides challenge response, its credentials containing authentication identity and optionally an authorization identity.
4. Server verifies the challenge responses, and identities and in particular, that the authentication identity is permitted to act with the given authorization identity. The server uses its own chosen internal mechanism for this purpose which is not dictated by the framework.
5. The server then responds with the outcome of the exchanges.

Some features that make SASL a superior choice to other frameworks include interoperability and flexibility as a result of the decoupling of mechanisms from protocols. It therefore becomes easier to programmatically opt for stronger authentications, mutual authentications and other requirements that a client can participate in [IMG+07].

A downside of the framework is that it does not specify any protection for the mechanism negotiation step and this could lead to downgrade attacks where the implementing system does not make other provisions to protect the negotiations [MZ06]. In such a case, an attacker can intercept and change the name of the agreed mechanism to a less secure one.

***Extensible Authentication Protocol (EAP)*:** EAP is a layered protocol framework functioning at the data link layer to provide network access authentication [ABV+04]. It allows different current and future authentication mechanisms (EAP methods) to be hosted in clients and servers as plug-ins. This capability makes it possible for peers to negotiate on an appropriate method such as EAP-TLS, PEAP-TLS, Kerberos, CHAP and many other possibilities during the authentication phase. It is therefore flexible to be used in devices with diverse capabilities and for different security requirements.

In its basic specification, EAP is made up of three entities — the *supplicant (peer)* requiring access into a network, the *authenticator* that grants access and an *authentication server (AS)* [San05]. The AS' role is to negotiate with a peer on the choices of authentication mechanism to use and also validates the peer's credentials before authorizing an authenticator to grant access. It is common for a single device such

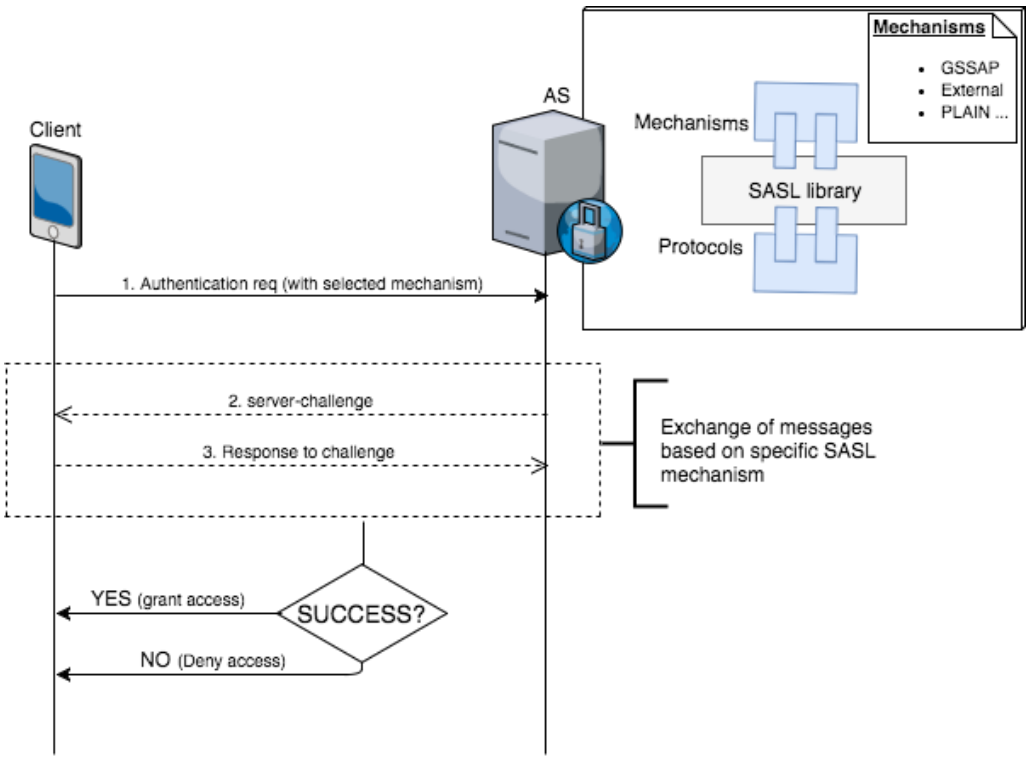

Figure 7: Message Exchanges In SASL Framework

as 802.11 AP to play the role of both authenticator and an AS but a typical scenario is a Remote Authentication Dial-In Service (RADIUS) implementing the AS.

The lowest level of this layered framework is the data link layer where a link establishment phase between the supplicant and the authenticator takes place. Links can be established using a serial Point-to-Point Protocol (PPP) connections or IEEE 802.1X-based access media. The second logical layer is the EAP layer where the core framework makes provisions for different authentication mechanisms to plugged in as EAP methods and at the very top is the authentication layer with the specific methods (TLS, CHAP, Kerberos) [CW05].

Authentication phase starts after the link is established and a request from either the peer or authenticator kicks in. The authenticator sends an identity request to peer and the peer responds with credentials. Based on the negotiated EAP method, a series of message exchanges between the three parties results in either a final success or failure message sent to a peer by the authenticator as shown in Figure 8. An advantage of this protocol is its support for different authentication mechanisms,

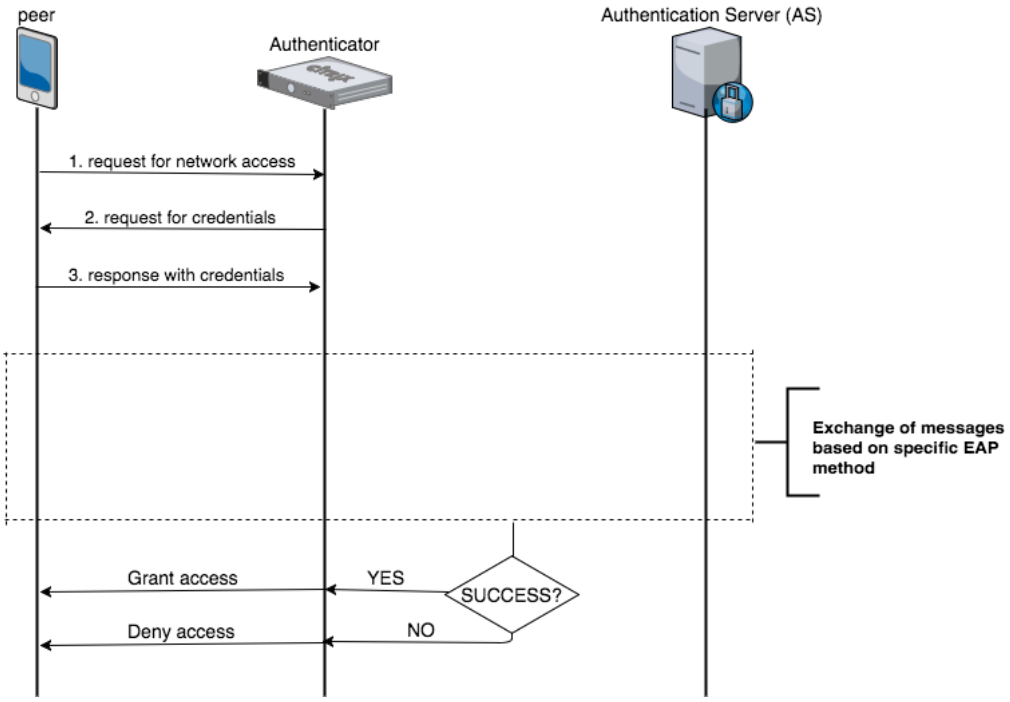

Figure 8: Message Exchanges In EAP Framework [San05]

making it possible for light weight protocols to be plugged in for use in the IoT environment. It could be implemented in a local hub using an access point and so well suited for protection of Device-to-Gateway interaction flows. One downside of EAP is its flexibility in accommodating several protocols makes it possible for some methods to allow clear text EAP conversations. This can expose a system to attacks by malicious users that have access to the link layer media [mal08].

***Internet Protocol Security (IPSec)*:** Working at the network layer, Internet Protocol Security (IPSec) [FK11] is a suite of protocols that provides network-level security for data exchange over Internet Protocol networks. It uses specific protocols such as *Authentication Header (AH)* for origin authentication, integrity, and replay protection for data packets whilst the *Encapsulated Security Payload (ESP)* is used to provide confidentiality and integrity.

IPSec can be configured to block or allow data traffic based on parameter like source and destination addresses, ports or protocol type. It is therefore useful in scenarios that require packet filtering. Through ESP, IPSec can also be used to restrict access to application server, hence good at securing back-end servers. The downside of

IPSec is its inability to secure multicast and broadcast traffic. Issues of incompatibilities might also arise when used in real-time communications that require *Internet Control Message Protocol (ICMP)* or *peer-to-peer* applications.

3.2 Authorization

This is the service that determines who is trusted for doing a given operation on an object or a resource in a system. There are usually different groups of resources and different kinds of operations applicable to these resources. The role of authorization is to establish whether an authenticated subject for an operation or request on an object is trusted for that operation on the given object [B.L91]. Authentication and Authorization services are two aspects that enforce *Access control* in a system — regulating fine grain control of resources [SPB+16].

Authorization can be done by attaching an access control list (ACL) to each object in a system. The ACL states which subjects are authorized for which operations. To enforce authorization, the service takes a subject, an ACL and an operation or set of rights and returns a "yes" or "no".

There are also other models for authorizing subjects in a system such as the *Role-Based Access Control (RBAC)*, *Attribute-Based Access Control (ABAC) model* and *Capability-Based Access Control (CaBAC)*. RBAC model groups available permissions into roles and users/subjects into different user groups with the same authorization privileges. Groups of permissions (roles) are then assigned to user groups during role-assigning tasks. Granting of access to resources in the system are enforced by the RBAC system based on the role assigned to a user. In effect, roles with permissions control what can be accessed. A down side of a basic RBAC model is that it gives pre-assigned set of permissions (through roles) to users and does not take into account dynamic attributes such as location of users and time of the day [KCW10]. Basic RBAC is therefore static in the provisions of permissions to uers. Additionally, there is the need for role-engineering prior to using RBAC. On the positive side, security auditing is simpler with RBAC because roles and their associated permissions are well understood. It also enables representation of large number of users with limited set of roles [CW13].

In ABAC, access is granted according to attributes presented by a subject hence attributes are the controlling factors in this model. It consists of two main aspects, *Policy rules* that specify the conditions under which access is granted. Second aspect

is *Architecture model/Enforcement component* that applies the policy. The model enables a dynamic acquisition of permissions at runtime depending on attributes presented by user to enforcement component. The downside of this is the need to understand and manage a potentially large number of attributes.

In CaBAC model, the concept of capability which is right granted to a subject in possession of it, is used. These capabilities usually come in the form of tokens, tickets or keys that give the possessor permission to access a resource [OMEO17].

In most implementations, it is common to see a combination of two or more of these models to achieve the desired results of the system. For instance, the Azure IoT hub uses both policy rules and tokens hence leveraging ABAC and CaBAC models respectively. The fusion of these two have several advantages. For instance the use of tokens or certificates in CaBAC ensures that fine-grained access control is granted at devices or platforms level. Another example of a combination of these models is in the XACML protocol that uses RBAC and ABAC. These protocols are discussed in details in this section.

Protection domains(Local and Global scopes): As part of the organizational structure of the access control model, there is a provision made for scoping of interactions between entities. The scoping introduces a logical boundary called protection domain around a set of entities that are assumed to know and trust each other within a certain local scope [Pap12]. Authentication is only required for entities that cross the boundary into a global scope. Some techniques employed by various runtime environments to enforces this include tools such as X.509 certificate and Kerberos service ticket.

In the following texts, we have a detailed discussion of some of the authorization protocols that can be used in the security framework. Table 2 gives a summary comparison of these protocols in terms of the main purpose they serve, their pros and cons and the data format they support.

eXtensible Access Control Markup Language (XACML): XACML is an OASIS[20] standard enforce attribute-based access control in systems. The core specification describes three main aspects namely *XACML reference architecture*, *XACML policy specification language* and *XACML request/response protocol* [XACML]. The policy language is used to declaratively express the access control rules which are combined by algorithms into policies mash-up. The XACML request/response protocol is used to exchange messages between the enforcement and decision making

[20] `https://www.oasis-open.org/`

components of the XACML specification.

One advantage of XACML is that it decouples the enforcement function from the decision and management functions so that authorization is implemented in depth at different layers. It also has extensions called *"profiles"* that exposes Authorization as light RESTful Web services with JSON encoded data. This enhances interoperability and also useful for constraint devices to participate.

As indicated in the beginning of the section, XACML is a combination of ABAC and RBAC models, and therefore gives the flexibility and advantages of ABAC such as use of dynamic attributes in granting access whilst maintaining RABC's advantages of easy auditing [CW13]. In the Web of IoT platforms, XACML is useful in implementing authorization at various cloud server where using other protocols like OpenId and LDAP to establish identities.

Some disadvantages include that it is a lot heavier that alternatives like OAuth and so mostly favourable for Web based applications.

XACML Reference Architecture The reference architecture specifies the standard required software components to deploy to effect enforcement of access control policies (see Figure 9).

- *Policy Enforcement Point (PEP):* This guards resources by intercepting client requests and sending them to a decision point.
- *Policy Decision Point (PDP):* The core of the architecture, responsible for loading policies and evaluating them against incoming authorization requests to eventually produce decisions for the use by the enforcement point.
- *Policy Retrieval Point (PRP):* Stores the XACML policies that decision point will load to make decisions
- *Policy Information Point:* Stores additional information about authorization context that may be needed by decision point. It could be database containing products information, user directories, LDAPs or active directory where attributes about the user accounts can be retrieved.
- *Policy Administration Point (PAP):* The point through which administrators can define and write authorization policies to be stored into the PRP.

XACML Core Data-flow As shown in Figure 9, data flows in XACML protocol consists of a series of message exchanges between the specification's components in

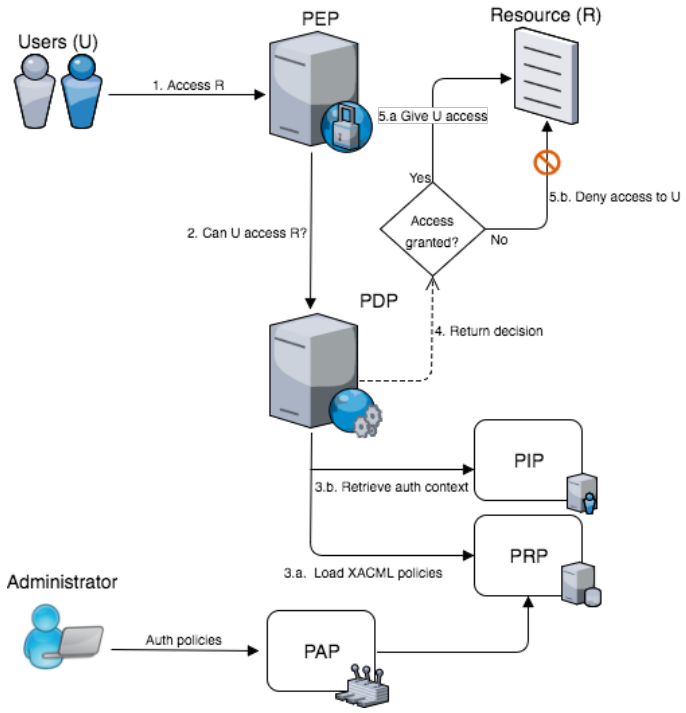

Figure 9: Message Exchanges In XACML

order to take decisions on authorization requests from users. The details of these exchanges are as follows:

1. User U sends access request for a document D.
2. PEP intercepts the request and sends authorization request to PDP.
3. PDP will load the XACML policies it is aware of from PRP and evaluates the incoming request against the policies. In the process, PDP might need to retrieve some attributes of the resource or user. For example if the policy is about the clearance level of the user compared to the document classification. In this case PDP will contact PIP for the user clearance level and document classification to make decision.
4. Decision is returned to PEP to let the request go through or deny.

XACML Profiles XACML profiles enable the extension of the core XACML specification to support specific needs such as exposing PDP authorization service as RESTful Web service. Though the core standards specify that request and response

messages between the PEP and PDP be formulated as attribute-value pairs using XML notation, it does not dictate the exact communication protocol to convey these messages. Most XACML implementations resorted to exposing the PDP authorization service as SOAP-based Web service with proprietary interfaces [Nai15]. This made it necessary to use vendor specific SDKs to implement the PEP service to be able to use the proprietary-interfaced services resulting in tight coupling and interoperability problems.

The REST Profile of XACML [RESTX] is one great solution to the tight coupling and interoperability issues by exposing PDP authorization services as RESTful Web services. It is easier to integrate REST with different languages including those that do not have great support for SOAP. There is also improved efficiency with the elimination of SOAP overheads such as verbose and bulky messages.

Another useful profile which helps to improve efficiency is the JSON Profile of XACML [JSONX]. This profile replaces the XML-encoded requests/response message formats between the PEP and PDP with light weight JSON encoded equivalents. This makes data parsing easier for constraint devices. The profile is used in conjunction with the REST Profile.

***Open Authorization (OAuth 2.0)*:** OAuth 2.0 [Har12] is an authorization framework that allows client application to get restricted access to hosted resources through an access token provided by an authorization server.

It uses HTTP protocol though other works [LST16], [FAKS14] have tried adapting it to protocols like COAP and MQTT 3.1. OAuth 2.0 protocol is simple and well suited for RESTFul design and JSON data. It allows for revocation of delegated permissions through token-refresh and expiration and promotes identity interoperability [OMEO17]. Identity interoperability will eliminate the need for a user to create an account for every service provider by enabling one provider to accept, trust and use an identity created and managed by another provider — a trust hub. Example is the case where Google or Facebook credentials are used to log into other platforms.

The OAuth 2.0 framework consists of basically four actors: The *Resource Owner (RO), Resource Server (RS), Authorization Server (AS)* and *Client*. The RO can be a human user or a device (in the case of IoT) that wants to access some resources from a resource-host server, the Resource Server. The user or RO might like to do this through an intermediary, say an app running on a phone. The intermediary in this case is the Client. The Authorization server is in charge of authorizing

the Client to access the resource on behalf of the Resource Owner/user. However, OAuth is not technically required to do authentication by itself and hence there may be need for a separate authentication server in the setup. Typically, flows in OAuth authentication process will consist of the following (see Figure 10):

1. The Client contacts Authorization server (Auth Server) for access to an API.
2. AS asks for user (RO) authentication. It typically delegates authentication to an Authentication Server (Auth Server).
3. Auth Server authenticates RO and informs AS.
4. AS then issues an Authorization token to the client.
5. Client subsequently uses token to access resources it is allowed to as contained in token from the Resource Server RS.
6. RS contacts AS for validation of token.
7. Upon successfull validation of token, RS responds client with the requested resource.

The access token (bearer token), which is a string, can be an JSON Web Token (JWT) which is plain text and so needs to be protected by a transport layer security. This is typically provided by TLS, hence OAuth has that requirement.

Some weaknesses of OAuth 2.0 include Interoperability problems — the many added extension points in the spec resulted in incompatible implementations. Hence one cannot be sure that a generic *Endpoint Discovery code* will work for the different implementations. For example, you may have to write separate pieces of code for Facebook, Google, Salesforce and so on. This issue is highlighted in a disclaimer in [Har12].

***User-Managed Access (UMA)*:** The User-Managed Access (UMA) [HMMC16] is a Web protocol built on top of OAuth that allows resource owners control access to all their protected resources hosted on different servers from a single point. It uses loosely coupled components (Authorization server, Resource server and Requesting party) to enable fine-grained access control and not just "consent of control" in the case of OAuth. The Authorization server authorizes access to any resource on the Resource server and servers as single point that owner can configure access

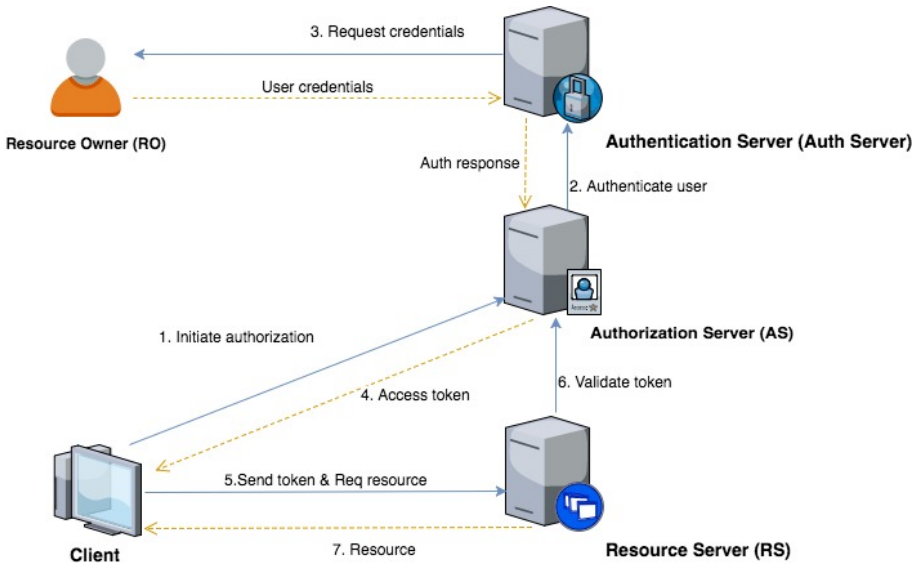

Figure 10: Message Exchanges in OAuth 2.0

control. Requesting party serves as a liability layer since resources will be shared with arbitrary third parties using delegation.

UMA is still in its draft stage though there are production platforms such as ForgeRock[21] that have already implemented its specifications and are providing services.

Protocol	**Data Format and protocols Supported**	**Main purpose**	**Pros**	**Cons**
OAuth	JSON, HTTP	API authorization between applications, used to obtain access tokens for Web APIs and protected resources	1) Supports Identity interoperability 2) Simple, Lightweight and well adapted for RESTFul design and JSON data, 3) Supports permissions revocation and runtime creation of an authorization context	1) Lack of implementation interoperability [OMEO17] 2) Only protects resources know at design time 3) No fine grain control of access
UMA	JSON, HTTP	fine-grained central access control management for resource owners	1) Fine-grained delegation and consent 2) Customizable 3) Specifies Protection APIs through which authorization server and resource server interacts	1) Still in draft stage and have not stood test of time yet 2) no wide spread infrastructure for its implementation yet
XACML	SAML, HTTP, SOAP, JSON	Standard language to express access policies and architecture to enforce policies	1) Decouples components 2) Easy security Auditing 3) Supports use of dynamic attributes 4) Supports JSON	1) Heavier than OAuth

[21] `https://www.forgerock.com`

JWT	HTTP, SOAP, JSON	Access Control and Authentication	1) Light weight 2) It is JSON and so easy to parse 3) can use asymmetric encryption 4) supports access scoping	

Table 2: Comparison of Authorization Protocols

3.2.1 Access Control with Security Tokens

These are usually opaque/encrypted string that contain claims to be exchanged between communicating entities and hence can be used to identify users or applications during API calls. There different proprietary tokens by vendors such as the *Shared Access Signature (SAS)* used by Microsoft in the Azure IoT hub. Another popular open source is JSON Web Token (JWT) [JBS15], a light weight object commonly used in Web applications for authentication and access control. The general structure of security tokens is the *metadata (header)* part that contains which type of token and what algorithm the token uses in its signature. The *payload* part contains the claims of the token that vouches for the carrier of that token. The signature part gives uniqueness to a token.

For the sake of this security framework, JWT is preferred to others such as Simple Web Tokens (SWT), SAS because it is less verbose than XML counterparts such as SAML tokens and hence easily parsed by constraint devices. JWT are self-contained because the payload can contain all the required data about a user, hence eliminating the need for extra queries to databases. JWT can be signed with asymmetric encryption methods using X.509 certificate unlike SWT that is limited to symmetric schemes with HMAC algorithm for signing. Another advantage of JWT is that the since the tokens are essentially JSON, with JSON parses common in most languages, it reduces the burden of developers on parsing the tokens.

Access tokens are an important part in controlling access of resources. One use case is to be able to control access to resource URIs by creating and sending tokens with restricted access to clients. This ability can further be used to scope resources and use appropriate access policies that contain the right permissions to sign the token as done in the Azure IoT hub using SAS tokens. For instance if an endpoint to access a shared resource by a device A is ***/devices/{device-A-Id}/sharedResouces***, we can use a shared access policy say, *GlobalPermissionsPolicy*, that contains global permissions for shared resources to sign the token. The same applies to restricted resources in local/private scope to a platform. This helps to get a scoped access

control to a platforms resources which is key to the security framework in this thesis.

Tokens (JWT) are also commonly used to authenticate clients. In this scenario, after first time login, a user is given an access token and subsequent requests will use this token to get authenticated without re-entering username/password.

3.3 Privacy and Confidentiality

Privacy in IoT enables a user to take control of his or her own data and hence decides who can be privy to which part of it [KVS16]. Techniques within privacy and confidentially services are aimed at making sure that messages are not available to unauthorized parties whilst in transit between platforms or from end nodes to platforms. IoT cuts across very critical systems like patient monitoring, traffic control and other mission critical systems. Therefore, there is bound to be flow of very personal and sensitive data which makes the privacy issue more compelling [SRGCP15].

3.4 Securing Remote Code Execution with Isolation Techniques

One of the goals for the Web of IoT platforms is to enable constraint devices to offload some computations to cloud or other platforms for execution. Code offloading is also important in the IoT as an enabler of edge computing where some data processing and analytics can be done at the edge of the network. There are associated benefits like reduced bandwidth costs (Section 2.2.5 has more on Edge interactions). Whilst offloading code is so important for the platforms, it also brings some security issues such as *preventing the offloaded code from accessing resources not exposed to it in the host platform* (Section 2.2.3). It would also be ideal to maintain the integrity, privacy and other relevant security metrics of the offloaded code in the foreign (host) environment. In this section, we look at an overview of Isolation techniques and how they can be leveraged in securing remote execution of code. To some extent, this will also provide secured environments for edge computing in IoT as edge/Fog nodes can safely execute code for others at the edge.

In order to mitigate security risks in multitasking systems, a fundamental step is to isolate running tasks from interfering with each others' execution paths as noted by Viswanathan et al. in [VN09]. Isolating tasks, especially for reasons of security, aims

at mitigating any possible harm that can happen in case there is any exploitable vulnerability.

There are several isolation mechanisms based on different criteria, such as enforcement location or isolation granularity, and some survey articles like [SWGI+16] and [VN09] have done extensive analysis on their requirements and performance. For instance, isolations are typically at done locations such as *(1) physical host, (2) hardware component, (3) Virtual Machine-based Isolation* and *(4) Sandbox-based Isolation.*

The main advantages of using isolation schemes in the security framework discussed in this thesis is to be able to support multi-tenancy in platforms. We envision that the devices at the edge of the respective IoT platforms/hubs will be able to interact locally in some scenarios, such as a more powerful gateway executing code for another gateway at edge of network (edge computing). We also envision that a given platform in the Web of Things platform/hub will be able to host other smaller IoT platforms/hubs. In order to enable this kind of cooperation, the services from different hubs should remain opaque to each other unless explicitly granted permissions to interact. Containers such as Docker are light-weight solutions to multi-tenancy, but if higher levels of separation are required, fall-back options include hypervisor-based virtual machines. Isolation will make it possible to securely execute foreign code in a secured, isolated environment and hence enabling computational offloading. This is needed in the Web of IoT security framework discussed in this thesis since it provides the environment for resource-constrained devices and platforms to outsource tasks they can not compute to other platforms.

Technique	**Implementation Examples**	**Code Requirement**	**Pros**	**Cons**
Hypervisor	VMware, Xen on ARM, VMware MVP, KVM	no	1) Full isolation 2) Acceptable performance overhead 3) Can host multiple kernels	1) Heavy weight 2) No flexibility on isolation levels 3) Hypervisors are single points of failure
Sandbox-based	MAPbox, chroot jail, SELinux, AppArmor	application source code or binaries modification required in some cases	1) Light weight 2) Flexible on isolation level	1) Sharing of kernel makes it less secure than VMs 2) No support for guest OS
Security Containers	Docker, Linux Containers (LXC)	no	1) Light weight 2) More efficient than some hypervisors 3) Easy to extend for billing	1) Not for hosting different kernels

Table 3: Comparison of Security Isolation Techniques

We next discuss the various isolation techniques in terms of the locations they can

be employed and the required implementation infrastructure as applied to IoT. A summary of this discussion is given in Table 3.

***Virtual Machine-based Isolation*:** Virtual machines are software abstractions that run on top of platforms (hardware or software) and expose the resources of the underlying platform in such a way that they can be consumed by any hosted software running withing the virtual machine. They play important roles such as resource optimizations, translations, replications and provision of isolation between multiple systems concurrently running on the same platform [SN05].

Depending on the level of abstraction, virtual machines come in different forms such as *process virtual machines, system virtual machines* and *hardware virtual machines.* Viswanathan et al. have done an extensive discussion of them in [SWGI+16]. For the purpose of the Web of IoT platforms, we will consider only system machines which do not require special hardware like the hardware virtual machines.

System virtual machine Isolation: Unlike process virtual machines where virtualization is at the Application Binary Interface (ABI) or libraries (API) level, system virtual machines provide all necessary hardware resources and ISA for the underlying platform. This is done through a piece of software called *Hypervisor or virtual machine monitor (VMM).*

A VMM is a piece of software that emulates an underlying hardware platform and exposes hardware resources as virtual ones to support and manage execution of virtual machines thereby enabling complete operating systems to run within it. VMM shares a platform's resources amongst the multiple guest systems and mediates access to these resources. By controlling critical paths to intercept and apply policies to monitored executions, mediation results in the necessary isolation between the virtual systems [VN09]. This isolation is strong enough that there is no sharing of resources between VMs than what is typically provided by networks between phyical machines [SPF+07].

Some advantages of hypervisor-based techniques include the ability to host multiple kernels, provision of administrative capabilities (root) within a VM as well as ability to save states of VMs, pause and resume later (checkpoint and resume).

There are two broad forms of hypervisors, namely Type-1 and Type-2. Type-1 sits directly on the hardware and exposes resources for a guest operating system, whilst Type-2 runs within a host operating system and in turn hosts guest operating systems [SWGI+16] (see Figure 11).

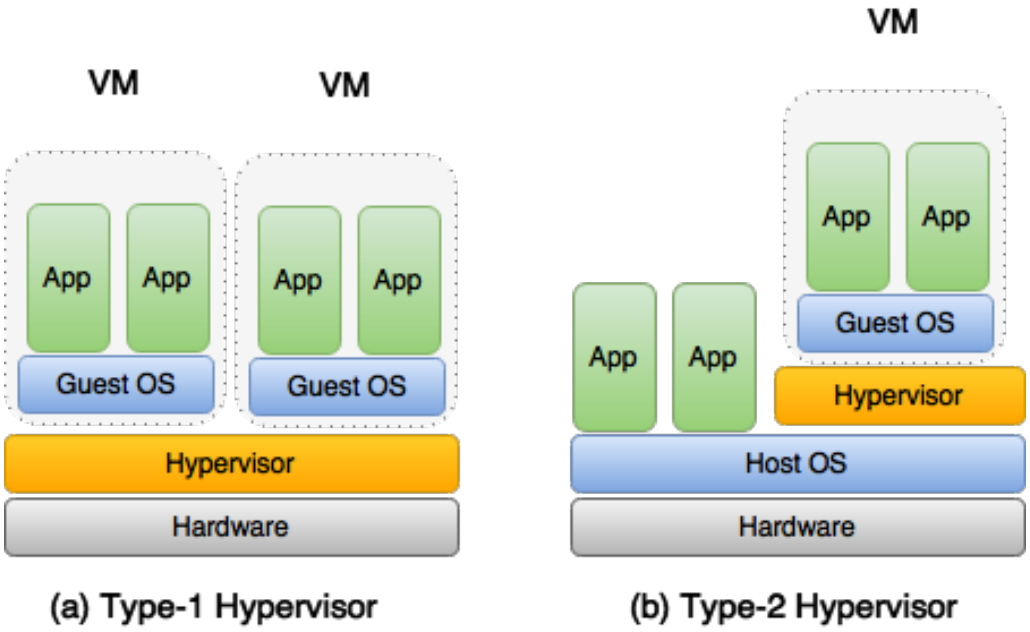

Figure 11: Hypervisor forms

Virtual machines have traditionally been implemented on resource-rich servers such as at the enterprise level and so could be used at the cloud hubs of IoT ecosystem. Whilst hypervisors have traditionally been popular with servers, optimization works such as *Xen on ARM* [HSH+08] and [BBD+10] have produced hypervisors capable of running on embedded systems such as the ARM processor. Toumassian et al. [TWS16] have done performance analysis on some of them with astounding performance by Jailhouse Partitioning hypervisor [Sin15]. These embedded level hypervisors makes it possible to also implement isolation at hub gateways in the context of IoT.

Sandbox-based Isolation: Maass et al. [MSCS16] give a broad definition of a sandbox in computing to be *"an encapsulation mechanism that is used to impose a security policy on software components"*. This broad view of a sandbox encompasses the role played by system virtual machines which we have already discussed. In this work, we restrict sandboxing to techniques smaller in scope than system virtual machines. They provide restricted environment for running some piece of untrusted code on a system by limiting its access to other external running applications or the underlying platform. In most implementations, sandboxes are implemented as layers between the isolated process and the operating system, making it a lighter

solution than system VMs [VBM15].

Based on techniques of restrictions, different categories of sandboxing are in common use. One such category is the Instruction Set Architecture (ISA) based where the mechanism restricts a program's activity at the level of the ISA. Typically, the ISA-based category uses binary rewriting to insert instructions at certain points of the application code that limits usafe operations (store or jump) within a target address space [WLAG93]. One disadvantage of this category is the need to modify source code or their binaries. Another issue with this is its dependence on the type of instruction set (RISC or CISC) as pointed out in [VN09] hence making it platform dependent. Examples of techniques in this category include the Software Fault Isolation in [WLAG93], Program Shepherding [KBA02] and PittSFIeld [MM06].

A second category of sanboxing is to provide restrictions to library or system calls at the level of the Application Binary Interface (ABI) through a configuration file. In some implementations of this category, restrictions are not only by individual applications basis but groups of applications with similar requirements. An example is the MAPbox technique described in [AR00]. Other examples include Consh [AKS98] and SLIC [GPRA98].

A third and more commonly used sandbox technique is the use of Access Control List (ACL) where the restriction is provided by explicitly attaching permission rules to resource accesses such as files, processes, networks and devices by programs. This is distinguished from the ABI-based approach in that the ACL-based is more generic and involves more than just the prevention of system calls as opposed to the case of ABI [SWGI+16]. Examples of ACL-based sandboxes include SELinux[22], UNIX chroot jail and AppArmor[23].

Container-based Isolation: Containers, just like most sandboxes, do virtualization at the OS level but they give much broader OS image than sandboxes. They provide shared virtualized OS image made up of *a root filesystem* and *a set of protected shared system libraries* [SPF+07].

Architecturally, infrastructure supporting containers are made up of *(a) the hosting platform* and the *(b) virtual platform or containers*. A hosting platform in turn comprises a shared OS image and administrative virtual machine to manage the containers. All the containers have their own private but identical view of the abstracted system made available by the hosting platform as shown in Figure 12.

[22] `https://selinuxproject.org`

[23] `http://wiki.apparmor.net/index.php/Main_Page`

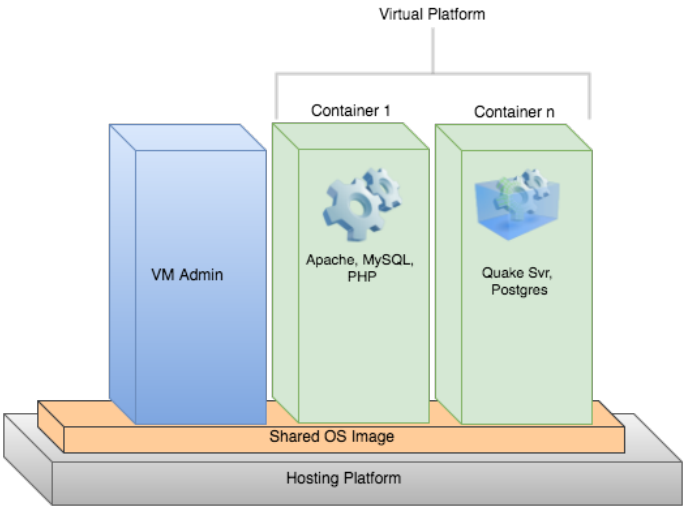

Figure 12: Isolation via containers

Guest applications run on the containers as they would have done on corresponding non-container based systems.

Security isolation in containers wraps around the use of internal OS objects such as PIDs, UIDs, shared memory and so on [SPF+07]. The basic restriction technique involves *(a) separation of namespaces or contexts* and *(b) application of access controls (filters)*. The restriction puts global system identifiers like System V IPC keys, PIDs and UIDs in different spaces. For example, each container corresponds one namespace. Objects in one namespace do not have references to those in different namespaces, hence abstracting the global identifiers as per-container global identifier. Filters are used at runtime to control container access to kernel objects.

Containers differ from hypervisors in the sense that though the latter also uses context and filter for isolation, it relies on hardware abstractions such as virtual memory address spaces, devices and privileged instructions instead of OS objects for the contextualization. Examples of real-world containers include Docker[24] and Linux Containers (LXC)[25].

One advantage of containers over other isolation methods is an increased system performance whilst guaranteeing a similar level of isolation in other VM systems. For instance, a comparative evaluation of efficiency between containers and hypervisors

[24] `https://www.docker.com/`
[25] `https://linuxcontainers.org/`

(VServer and Xen) by Soltesz et al. [SPF+07] shows that container-based techniques give up to twice the performance of hypervisors due to I/O overheads in the latter.

4 The Security Framework for the Web of IoT Platform

To address the security and privacy needs of the highly diverse IoT environment, a comprehensive framework is required. Such a framework should be able to provide a fine-grained protection of the platforms' data at all levels of interactions and at the same time be open to extension. This section presents the Web of Things Security Framework, a modular, extensible and fine-grain security framework for the integrated Web of things platform. It provides the authentication, access control and privacy needs that will ensure the protection of the platforms' data whilst making it possible for controlled sharing of resources between participating platforms.

The framework uses selected state-of-the-art protocols discussed in Section 3. We base the selection criteria of protocols on the unique characteristics of this environment such as low computational power of end-devices/sensors, diverse nature of devices, the nature and quantities of data produced and so on. In that regard, there may be several eligible protocols that address the platforms' authentication, access control and other security needs, but the choice will fall on what type of interaction we are securing and at what level of the platform (edge, gateway or cloud) the interaction is taking place. For instance, when the interaction requires authentication and is at the edge of the platform's network, eligible authentication protocols will be those that use relatively low computational power since edge devices closer to the edge such as micro-controllers have relatively lower computational power than servers at gateways or in the cloud.

4.1 Categorization of Interactions for Protection

To ensure granular and fine-level application of security protocols to the platform's interactions, the security framework categorizes devices and interactions within the platform into vertical and horizontal dimensions (see Figure 13). The horizontal dimension has subgroups of three labeled as Level 1, Level 2 and Level 3.

1. **Level 1 interactions:** This level is made up of edge devices like sensors, and

communication model is Device-to-Device. The interactions at this level are labeled H1 as seen in Figure 13. These are the kind of interactions we envision in the Web of IoT platforms where devices at the edge of the network will be able to communicate with each other directly without a bridge. For instance, when domestic appliances like fridges and washers are able to communicate directly in meaningful ways without an intermediary.

2. **Level 2 interactions:** This second level is made up of interactions (H2) between different gateways. The H2 interactions are the interactions that take place in Platform-to-Platform pattern discussed in Section 2.2.4. The gateways are assumed to belong to different platforms.

3. **Level 3 interactions:** The topmost level is H3 interactions which comprise of flows between cloud-hosted services or cloud-based platforms. These flows are modeled by the platform-to-platform pattern, but in this case, the platforms are cloud-based.

The vertical dimension is made up of interactions (cross-level) that take place between devices belonging to the different levels in the horizontal dimension. Consequently, we have three categories in this domain based on which pair of levels the interactions traverse. They are as follows:

1. **V1 interactions:** Devices here include sensors with embedded micro-controllers such as Nest cameras and thermostats, or detachable sensors that can used together with multipurpose micro-controllers like Arduino boards and Raspberry Pi. Protocols often used include Bluetooth, MQTT and wifi. Authentication and authorization protocols are usually intrinsically built into these protocols such as the cryptographic key-based authentication for Bluetooth.

2. **V2 interactions:** In these kind of interactions, gateways perform a leading role in ensuring that messages are able to be exchanged between end devices and the IoT hubs. A gateway provides bridge between the edge devices/sensors and the various IoT hub end-points. They perform various functions such as protocol translation, running edge analytics, making time-sensitive decisions to reduce latencies, security and privacy enforcement. These added roles differentiate them from simple routing devices such as NAT devices. Gateways come as either *protocol gateways* that are deployed in the cloud or *field gateways* deployed locally such as micro-controllers. In some IoT such as the Azure

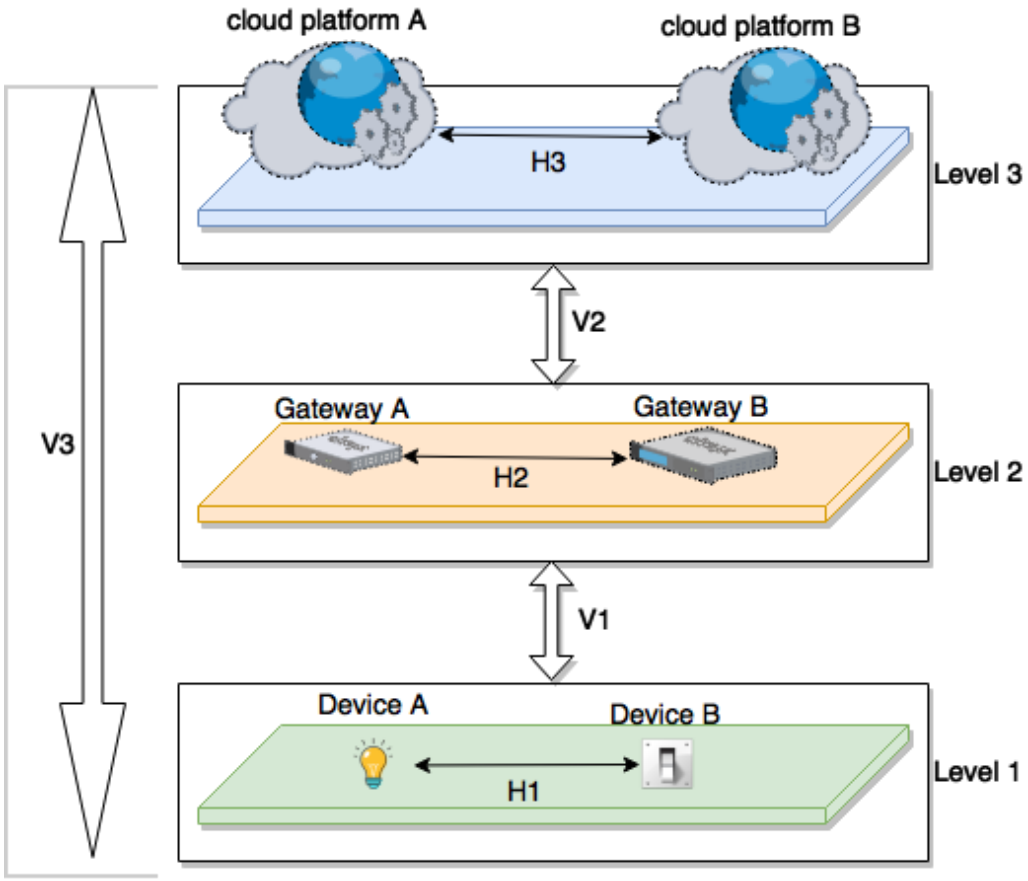

Figure 13: Categorised interactions for security framework

IoT platform, both types of gateways are used depending on the needs of the end user.

The interactions can be between field gateways in different hubs or from a field gateway to cloud-based hub. In the Web of IoT platforms, we envision these interactions to be enabled by RESTful APIs and consequently the security goals in this category of interactions will be how to secure the APIs.

3. **V3 interactions:** In this category of interactions, devices that are capable of directly connecting to the Internet using HTTP protocol can directly interact with cloud IoT hubs. For instance, a user can connect to a cloud hub through a smart phone without a bridging gateway. In this case, we don't require gateways for protocol translations, and so the goal in securing these kinds of interactions will be how to to secure RESTful APIs through which these

interactions take place.

4.2 The Security Framework's Architecture

In this section, we look at how the various security considerations (authentication, access control and privacy) are handled by the security framework. Use of security containers to secure edge computing in the Web of IoT platforms is discussed in Section 3.4. We start by exploring a modular approach to handling authentication in the framework. This is followed by a handling of access control through *OAuth 2.0 and identity registry* and concludes with handling of secured executions through isolation containers.

The architecture of the security framework is organized into three main subsystems. The *Authentication subsystem* takes care of the authentication of devices, gateways and platforms/hubs. It is mainly made up of the modular authentication structure discussed in Section 4.2.1. It should be noted that the modular approach for authentication does not apply to H1 interactions. H1 interactions are envision to be direct device-to-device interactions at the edge of the network. We expect authentications in these interactions to be handled by the communication protocols they use such as Bluetooth or Near-field communication (NFC).

The second part of the framework is the *Authorization subsystem*. This is made up of the protocols and mechanisms that authorize the use of resources in the Web of IoT platforms. It applies to V1, V2, V3, H2 and H3 interactions. Again H1 is left for the underlying communication protocol to handle just like the case of authentication.

The third subsystem is that of the *Secure Execution Subsystem*. The role of this component is to provide secured environments within gateways or platforms (local or cloud) to execute foreign code. Foreign code in the context of this framework is any offloaded code from a device or platform that seeks computational services from a different device, gateway, hub or platform.

4.2.1 Handling Authentication with Security Modules

A key design goal of the framework is to make it easily extensible and at the same time to secure interactions in the Web of IoT platforms at fine-grained levels. To that extent, we propose organizing the solutions (security protocols) for the identified security requirements in the Web of IoT platforms into modular security plug-ins.

This modular pattern is inspired by the FRESCO framework [SS13], a security application development framework that enables development of security services in OpenFlow-enabled networks as composable modules. We then leverage behavioral software design patterns such as the *Chain of Responsibility (COR)* [GHJV95] to link these modules together into a dynamic authentication subsystem for the security framework.

One advantage of this approach on the authentication part is that the security framework is able to offer several different authentication mechanisms that are supported by the diverse IoT platforms taking part in the Web of IoT platform. For instance, if an IoT platform say A supports only certificate-based authentication, it can be handled by the appropriate certificate-based module residing in the security framework. When a different platform say B that supports only username/password with tokens seeks authentication, the request will also be picked up by a different module that can handle it.

Through the *Chain of Responsibility (COR)* pattern, we are able to chain together the different authentication modules into an autonomous orchestration that can dynamically switch between modules. With this kind of orchestration, there is no extra messaging between the IoT platforms and the security framework to negotiate on the type of protocol to use for authentication.

Another good aspect of the the modular design is that the *COR* pattern makes it easy to extend the chain by adding concrete handling objects as will be discussed shortly. This behaviour enables the ability of the security framework to be easily extended to accommodate future mechanisms.

We next discuss the structure of the modules, which are the building blocks of authentication in the security framework and how the COR pattern can be leveraged to enable a dynamic, modular and extensible approach to handling authentication in the framework.

Architecture of the Modules: We propose implementation structure of each of the modules to follow what is described for the concrete handler objects in the COR discussed in [GHJV95]. In this context, we use modules and class interchangeably. Modules correspond to the building blocks of our modular authentication subsystem of the framework. A module's implementation can be a simple class in any object oriented programming language or an interface that serves as a facade to a subsystem. However, in this work, implementations of the modules are simple classes, hence module A will be implemented as a concrete class with name A. Structurally, a

module is an AuthenticationHandler class with an operation to authenticate clients and an attribute that points to its successor. The successor is also of type AuthenticationHandler. AuthenticationHandler class is a parent for all candidate handler classes (modules).

ConcreteAuthenticationHandler is a concrete class that inherits the AuthenticationHandler class. Each of the modules in the chained authentication structure is implemented as a ConcreteAuthenticationHandler. This class handlers authentication request is responsible for using its *authenticate* operation. The class can also access its *successor*. Figure 14 shows the structure of the implementation in a UML class diagram notation.

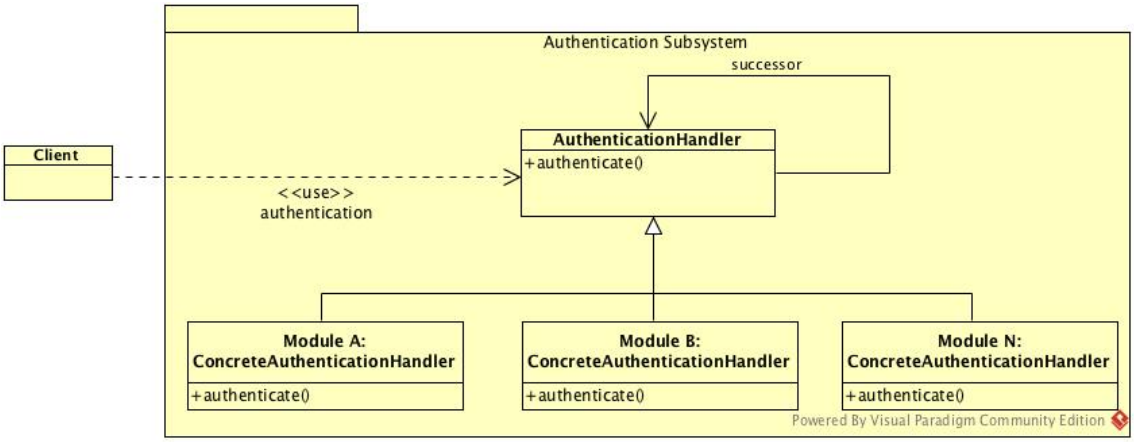

Figure 14: Modules in Authentication subsystem

Chaining Modules with COR pattern:

The COR is a design pattern in object oriented software development whose basic intent is to provide decoupling between interacting components and make it flexible for a client component to have potentially many server components that will give service to the client at runtime [GHJV95]. The basic idea is to model a client as an object that needs service from another object, say server. We don't want the client to be tied to a particular object that will handle its request at runtime, instead we model all potential handlers of the client's request as objects and chain these handler objects together. With this arrangement, when a request comes from the sender, it is passed along the chained handlers until an appropriate handler handles it. One problem that needs to be overcome before we can enable the security framework

to dynamically replace different modules to handle different authentication schemes from the diverse IoT platforms is as follows: If a handler module A is not able to handle a given authentication request, it should pass it on until the appropriate handler receives the request. In this case the module that ultimately authenticates a client is not explicitly known to the client. This kind of problem is is common in object oriented software development, and a commonly used design that solves it is the COR design pattern [GHJV95]. We leverage this kind of orchestration in the security framework where our handler objects are the composable modules. These modules encapsulate different authentication protocols but provide the same interface. The modules are able to pass on requests to other modules (successors) until an appropriate module with the right authentication scheme authenticates the client. The set up gives multiple authentication modules the chance to authenticate a client. It also gives us a chance to add a default module which handles a given authentication request when there is no module with an appropriate scheme to do so.

Passing authentication requests along chained modules: In order to be able to pass authentication requests along the chain, each of the modules shares common interface for handling requests and for accessing its *successor*. We define an *AuthenticationHandler* module which is essentially a class in an object oriented language. AuthenticationHandler has operation *authenticate* that takes *request* object as an argument. The request object contains necessary information on what kind of authentication scheme the client has sent and credentials to use that scheme. With this kind of information, each of the chained module handlers can determine whether it can handle the incoming authentication request or pass it on to its successor. For instance, when the request comes in and module A picks it initially, it checks the request object to verify whether it should pass it on to module B or not. If it decides to pass it on to B, the process gets repeated until appropriate handler handles it or it ends up at the default handler which is the last module in the chain (see Figure 15).

Adding new modules:

The arrangement also makes it easy to add new modules; since we know that at each time in the framework the last handler is the default handler, when we want to add a new module the task involves inserting it into the chain and making it the successor of the module before the last module. We finally make the default module the successor of the inserted module. This follows the pattern of inserting

an element into a linked list.

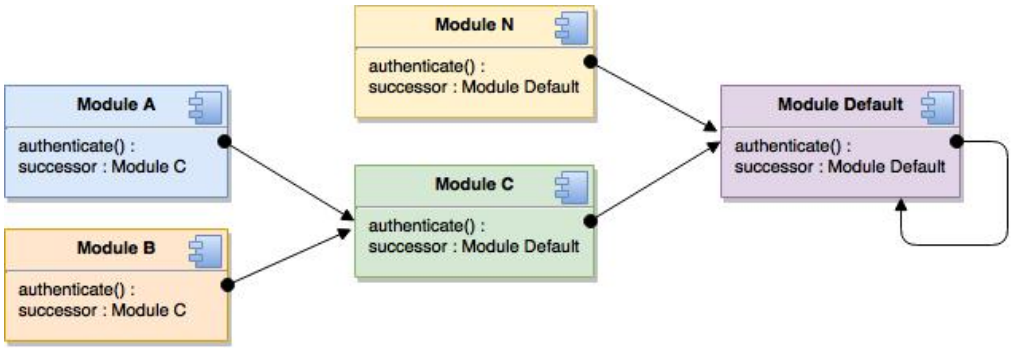

Figure 15: Chained security modules

Selecting Authentication protocols to compose modules Based on discussions of the security protocols in Section 3.1, we provide the following guide to building an authentication module for the framework.

1. *X.509 Specification based Certificates:* To compose certificate-based module, X.509 certificates are a good choice since it is nonproprietary and supported by popular protocols like TLS/SSL, EAP-TLS and LDAP. Therefore, a module using this certificate can handle authentications using the above protocols with little tinkering. The client (IoT platform) can use an existing X.509 certificate associated with it or a self-generated and self-signed certificate using tools like OpenSSL. A certificate can also be a CA-signed X.509 certificate obtained from a Certification Authority (CA).

2. *TLS with Client Certificate and Token Authentication:* This choice leverages the benefits of client certificate and token authentication as discussed in Section 3.1.1. The JSON Web Token (JWT) has wide spread adoption and so would be good fit.

3. *OAuth spec for TLS client authentication with X.509 certificate:*A module composed with these protocols will let client register for TLS authentication with OAuth 2.0 server by providing details of the X.509 certificate it will use. The client can send request with flags for TLS authentication using the X.509 certificate. At the OAuth service, a token is issued. This module will ensure that at the protected resource server, client submits its acquired token whilst authenticating with its certificate using TLS.

4. *SASL:* An example of the framework-based authentication mechanisms that supports application level protocols such as TLS.

4.2.2 Bootstrapping Security with Hardware Cryptoprocessor

Software-based security solutions are still proned to vulnerabilities if the software platforms hosting such solutions are compromised. To tackle this issue, there is the need to establish a Trusted Computing Base (TCB) in our authentication process using hardware-rooted trust. The hard-to-alter characteristics of hardware makes them more difficult to be modified by adversaries. We therefore employ the use of hardware chips for bootstrapping security in our edge devices like the gateways and servers. Establishing a TCB in these devices will secure the process from bottom up by ensuring the integrity of the certificates used in establishing sessions among the edge devices and between edge devices and servers. There are several alternatives such as the TrustZone, M-Shield, SmartCards, Trusted Platform Module (TPM), etc. [EKA14]. However, we will use TPM whose specification are open and it is also platform generic unlike others such as Trustzone.

Trusted Platform Module(TPM): This is an example of external secure element architecture where dedicated chips are generally employed in the implementation. It enhances hardware platform security by ensuring that the device is securely booted into a trusted state. TPM also provides a mechanism to record system measurements in a manner that they can't be tampered with. These measurements will be stored in shielded locations. TPM can then securely report these measurements (platform state) to a third party, hence, enabling remote attestations. For the framework, modern servers have TPM chips built into their mother boards that can be leveraged. We also employ ad-on TPM boards such as the Infineon OPTIGA™ TPM on the devices (Raspberry Pi, Arduino, etc.) that don't come with built-in modules. By using TPM, we are able to boot our devices into trusted states, protect secrete keys for the devices in secured locations and perform encryptions within the TPM modules.

4.2.3 Handling Access Control with OAuth 2.0, Device Identity Registry and Access Tokens

In the authorization architecture of the framework, we are going to have access control policies. A policy is made up of a set of permissions to perform certain

operations on resources stored in the resource server of the Web of IoT platform.

In our implementation, we provision an identity registry similar to that provided by the Azure IoT hub, but in this case, the registry can contain either a whole IoT platform/hub credentials or a single IoT device's credentials. By making it possible to have entries for both individual devices and platforms/hubs, we are able to grant permissions at per-device level as well as hub-level. For instance, it will be possible to grant a set of permissions (using a given policy) to say all devices connected to a given hub H. A second advantage is that we make the registry generic to be used at a hub level (including gateways) for making authorization decisions at the edge of networks thereby reducing bandwidth requirements and time for responsiveness.

Access control for Level 1 and 2 interactions: For granting access to resources at levels 1 and 2 of the categorized interactions, we employ the use of tokens (JWT). As depicted in the Figure 13, interactions at this level include H1, V1 and H2. The devices at these levels such as gateways or micro-controllers are usually constrained in resources and hence inappropriate for relatively resource-demanding protocols like XACML. OAuth 2.0 can suffice in some cases but will require extra message exchanges with Authorization servers. However, since most exchanges at these levels will have to do with devices within the local hubs, the risks of external adversaries are reduced. We will therefore employ JWT in authorizing devices to resources stored in gateways in Level 2 or those stored right at the edge of the networks in Level 1. There is also a possibility of proprietary protocols for some devices at level 1.

Access control for Level 2 and 3 interactions

The interactions at the levels are usually between gateways and cloud platforms or cloud-platform to another cloud-platform. At these levels, the devices are mostly well resourced servers that can store lots of data and perform complex computations. This is especially the case for H3 interactions. The choice of protocol to support authorization of resources for these interactions is the OAuth 2.0 protocol. The reasons for chosing OAuth 2.0 are as follows: For one, through the use of tokens, OAuth enables the sharing of resources without exposing user's identity or credentials. Secondly, through the use of *Proof of Possessions* and *Disconnected Flows* [Kri17], OAuth can be engineered to still support authorizations in times of network disruptions. The protocol is also in popular use by giants such as Google.

Since OAuth 2.0 does not directly implement authentication, we will employ Authentication servers in the setup. The authentication server will be in charge of au-

thenticating users through the modular approach discussed in Section 4.2.1. Though protocols such as OpenId and UMA do provide user identification on top of OAuth 2.0, they will not suffice in the case of the security framework which needs to be able to support different authentication schemes dynamically at runtime.

In the security framework's implementation, the *identity registry* is hosted by the Authentication Server. As discussed earlier, the registry's entries are credentials for either platforms or end devices. Eligible credentials correspond to the authentication methods supported by the framework in the modular authentication subsystem. It is against these credentials that a user is authenticated and subsequently the Authorization server grants tokens for a client to access resources on behalf of the user. Details of these interactions are shown in Figure 10.

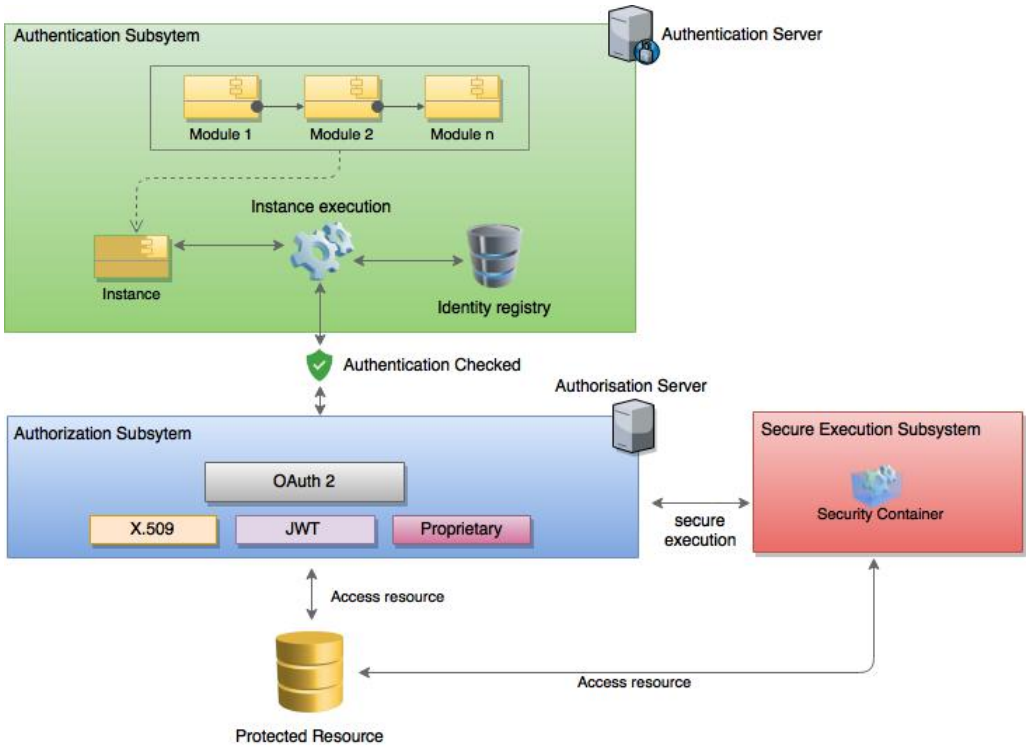

Figure 16: The Security Framework

4.2.4 Secured Edge computing with Security Containers

The security framework employs the use of security containers as discussed in Section 3.4 to support secured executions of foreign code in platforms/hubs. Foreign

code in the context of this framework is any offloaded code from a device or platform that seeks computational services from a different device, gateway, hub or platform. The security framework supports the creation of instances of containers at run-time, either within the gateways in Level 2 of the categorized interactions (Figure 13) or at Level 3. For the purpose of this thesis, we mostly mention the use of containers at the edge of the network (level 2) interactions. Using containers at higher levels such as within servers in the cloud is implicit since they are more resourced than edge devices. In order to edge computing in the Web of IoT platforms, the security framework leverages the container-based isolation technique . The container infrastructure will enable devices at the edge to securely offload computations they can't support to neighboring devices or gateways. There are a number of reasons for the choice to use containers at the edge level one of which is performance time as compared to virtual machines(VM). More significantly, containers are light weight as compared to other isolation techniques like VMs (Section 3.4). The container technology of choice is Docker because it is built on LXC and hence is a lightweight, portable, self-sufficient LXC container that can run virtually anywhere.

To use docker, we first need deliver it to the gateways by installing the docker image. There are also products like *resin.IO* [26] whose work can be leverage to deliver docker to edge devices with different architectures.

The basic architecture of the edge computation isolation in this framework follows the techniques used in the cloud-based compiler called *CodePad*[27]. CodePad is itself a product of the *CombileBox*[28] that provides a sandbox to run untrusted code.

The secure execution subsystem of our framework exposes an API that clients will call to offload their computation. This API points to a supervisor (typically shell script). The duty of this supervisor script is to create a new instance of the docker container with all the needed resources to service client calls. When the new docker instance is started within a gateway, the supervisor transfers the code to the instance where compilation and the execution happens in the confines of the container.

The output of execution will be read to a file from whence the supervisor can read the content and returns it to the client. See the diagrammatic description of the process in Figure 17. By employing the docker container, all activities of the foreign code are restricted to only resources explicitly made available to it and hence reducing

[26] `https://resin.io`

[27] `https://codepad.remoteinterview.io/IMJPDHDFCQ`

[28] `https://github.com/remoteinterview/compilebox`

unintended exploits on the host gateway. With this set up, constraint devices are able to leverage edge computing to accomplish tasks they would not have been able to do otherwise.

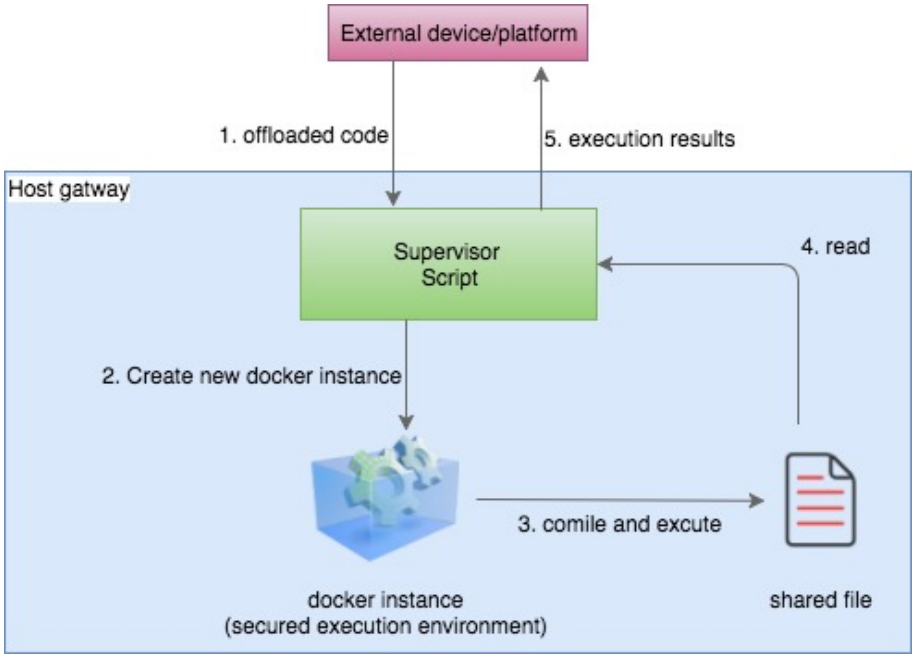

Figure 17: Secure Execution Subsystem

5 Discussions

The goal of this thesis work was to design a comprehensive security framework for the envisaged Web of IoT platforms which will interconnect the numerous non-interoperable IoT platforms existing today. Security requirements that will support such a Web of interconnected platforms were discovered to include authentication, authorization, privacy and confidentiality and secured environments for code execution. In order to handle these requirements at fine-grained levels, we categorized the main interactions that need protection in the Web of IoT platforms into vertical and horizontal dimensions. For horizontal dimension interactions, there are three levels. Devices at the edge of the various platforms generate data and also perform actuation jobs. They make up Level 1 of the horizontal interactions. We label the interactions within this level *H1 interactions*.

Level 2 of the horizontal dimension is made up of gateways which could be as simple as single-board computers such as Arduino boards. The end devices are mostly connected to them and interactions within the level is marked H2.

The topmost layer, Level 3 is mostly the cloud layer to which most IoT platforms transport their data for purpose of analytics, storage, etc. Interactions are labeled as H3 interactions.

The Vertical dimension is made up of all the interactions that cut across these levels.

The main architecture of the security framework that provides the needed security requirements include the three main subsystems (Authentication, Authorization, and Secure Execution). In this section, we give summary discussions for each of these subsystems with regard to the choice of technologies used and implementation considerations.

5.1 Authentication Subsystem

For authentication, our main approach in the security framework is to use a modular approach. The main reason behind this approach is because of the diversity in terms of protocol technologies used by the various devices and platforms. The goal is to make it as open as possible for the authentication process to support different factors of authentication such as what a subject has, knows or is made up of. The modular approach therefore makes it possible for simultaneous support for certificate based, tokens, username/passwords etc. Another desired consequence of the modular approach is the ability of the Authentication subsystem to easily add on new authentication schemes that come up by simply creating modules that encapsulate these new schemes and implementing standard interfaces. This is made possible by leveraging the *Chain of Responsibility (COR)* design pattern in object oriented software development. Thus, not only does the modular approach make it easy to accommodate diverse authentication schemes, it also makes the framework very extensible to future changes. Implementation considerations for the modular approach will include the use of object oriented programming language as a result of adapting the COR pattern. It should also be noted the authentication subsystem leaves out authentication for H1 interactions to the basic protocols that enable communication for those interactions. Hence, if devices use Bluetooth, it is expected that bluetooth will provide it.

For the choice of specific protocols, we used X.509 certificates standard because

of their popular usage in existing systems. There are also several implementing libraries in most programming languages. We settled on JSON Web tokens which are very light and very popular too. The ubiquity of javascript also makes it very easy to find libraries that will easily parse JWT. It should be noted that the other protocols discussed for authentications such as Kerberos and Shiboleth can all be supported with little modifications in some cases when this modular approach is implemented.

5.2 Authorization Subsystem

For this subsystem, leading protocol candidates included XACML, OAuth 2.0 and its profiles such as UMA and OpenID. However, XACML is heavier in terms of computational resources and storage required. OAuth is lighter coupled with its popularity in the Web and support from big giants like Google makes it a better choice. OAuth also supports revocation of permissions at runtime which is a good thing for the IoT ecosystem.

Implementation considerations include the provision of authentication setup since the OAuth 2.0 specification does not directly take care of that. However, we fall back on our modular authentication subsystem. Another issue is, since we are using this modular authentication approach, we don't want to be bounded by which authentication protocols to use, hence the decision to go for the core OAuth 2.0 protocol instead of its profiles like UMA. Another consideration is that we still make it possible to use other authorization protocols at some levels such as the use of certificates in level 1 interactions.

5.3 Secure Execution Subsystem

A desirable feature of the Web of IoT platforms is the provision of enabling environment for edge computing and computational offloading. Edge computing, which is not yet popular in IoT is necessary to reduce bandwidth requirements. This is especially important considering the large amounts of data generated in IoT ecosystem. The framework makes use of Docker containers to provide secured execution environment at gateways to support edge computing. Enabling secured execution environment at the edge is by no means all that is required to secure edge computing in IoT. However, it is a good start and there are some recommendations in the conclusion section of this thesis.

Secured execution is not only an enabler of edge computing but also a necessity for general computational offloading in the Web of IoT platforms. Since platforms vary greatly in their resource capabilities, its desirable for a less resourced platform say P1, to be able to tap into the capabilities of a well resourced one P2. However, just like the case of code offloading at the edge, we also need to secure offloaded code at the platforms. The good thing about securing execution at levels above the edge is that servers can support other forms of isolation techniques such as VMs. But all the same containers are found to be very useful in securing execution of offloaded code.

Implementation considerations include the choice of docker as our security container. Docker is light-weight with almost no start up penalty. It can carry any payload and its associated dependencies and can run in different platforms. Due to the resource constraint nature of most gateways, we use only containers to provide needed isolation at the edge. However, the framework is opened to other forms of isolations such as VMs or hardware techniques that are deemed necessary.

6 Conclusion

This thesis work presented a security framework that will provide the security needs of an envisaged Web IoT Platforms. This was done by first looking at the various interaction patterns that are present in the current IoT platforms and envisage interactions that will be needed in the Web of IoT platforms. The need for this was to make sure that we have a comprehensive understanding of what flows need protection in the platforms. We went further to give detailed discussions of some state-of-the-art security protocols in use today with special attention on how to leverage them for the security framework. The architecture of the security framework was finally presented using selected protocols discussed and the Chain of Responsibility design pattern.

In designing the framework, we came up with three main subsystems to handle authentication, authorization and secure execution of code respectively. The goal in designing the authentication subsystem was to make it possible for different authentication schemes to be used in the framework. This is because the IoT ecosystem is so varied in their choice of technologies including that of security mechanisms. Hence, there is the need to accommodate different protocols as much as possible. It is also important to have an extensible feature that will accommodate future

authentication protocols. This goal was achieved by encapsulating authentication protocols as modules and chaining using the Chain of Responsibility design pattern. We argue that this modular pattern does not limit us to some set of authentication protocols and so did not go specific on selecting a subset of the authentication protocols. Different protocols get encapsulated into modules that will adhere to specific interface and be added to the framework.

We however chose OAuth 2.0 for its lightweight, ubiquitous and other properties over computing protocols. The same went for using docker container for isolating execution of foreign code.

The framework as provided in this thesis by no means provides a bulletproof solution. There are bound to be future enhancements, and some of the recommendations at this stage include the following: Firstly, there is a need to provide privacy and confidentially over any offloaded code, be it at the edge or servers. We therefore recommend that future works look at measures such as fully *Fully homomorphic encryption schemes* that will make it possible for code to be executed in its encrypted form. This will provide the needed privacy for an offloaded code. Secondly, we recommend the exploration of data annonymization techniques such as K-anonymity in the effort of providing confidentiality within the framework.

In conclusion, the contribution of this thesis is the provision of the comprehensive security framework to provide the security needs of the Web of IoT platforms, and this was achieved using state-of-art security protocols.

References

ABV+04 Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J. and Levkowetz, H., Extensible authentication protocol (eap). RFC 3748, RFC Editor, June 2004. URL `http://www.rfc-editor.org/rfc/rfc3748.txt`. `http://www.rfc-editor.org/rfc/rfc3748.txt`.

AJR11 Al-Janabi, S. T. F. and Rasheed, M. A.-s., Public-key cryptography enabled kerberos authentication. *Developments in E-systems Engineering (DeSE), 2011*. IEEE, 2011, pages 209–214.

AKS98 Alexandrov, A., Kmiec, P. and Schauser, K., Consh: Confined execution environment for internet computations, Dec 1998.

AR00 Acharya, A. and Raje, M., Mapbox: Using parameterized behavior classes to confine untrusted applications. *Proceedings of the 9th Conference on USENIX Security Symposium - Volume 9*, SSYM'00, Berkeley, CA, USA, 2000, USENIX Association, pages 1–1, URL `http://dl.acm.org/citation.cfm?id=1251306.1251307`.

AS16 Aggarwal, C. and Srivastava, K., Securing iot devices using sdn and edge computing. *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*, Oct 2016, pages 877–882.

BBD+10 Barr, K., Bungale, P., Deasy, S., Gyuris, V., Hung, P., Newell, C., Tuch, H. and Zoppis, B., The vmware mobile virtualization platform: Is that a hypervisor in your pocket? *SIGOPS Oper. Syst. Rev.*, 44,4(2010), pages 124–135. URL `http://doi.acm.org/10.1145/1899928.1899945`.

B.L91 B.Lampson, *Computers at risk : safe computing in the information age*. National Academy Press, Washington, D.C, 1991.

BL14 Blackstock, M. and Lea, R., Iot interoperability: A hub-based approach. *Internet of Things (IOT), 2014 International Conference on the*. IEEE, 2014, pages 79–84.

JSONX Brossard, D., Json profile of xacml 3.0 version 1.0. edited by david brossard. 11 december 2014. oasis committee specification 01, December 2014. `http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/cs01/xacml-json-http-v1.0-cs01.html` Latest version: `http://docs.oasis-open.org/xacml/xacml-json-http/v1.0/xacml-json-http-v1.0.html`

BTV15 Blazquez, A., Tsiatsis, V. and Vandikas, K., Performance evaluation of openid connect for an iot information marketplace. *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*, May 2015, pages 1–6.

CDM03 Crall, C., Danseglio, M. and Mowers, D., Ssl/tls in windows server 2003. *Microsoft Corporation publication*, 31.

CS05 Cantor, S. and SCAVO, T., Shibboleth architecture. *Protocols and Profiles*, 10, page 16.

CSF+08 Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R. and Polk, W., Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC 5280, RFC Editor, May 2008. URL http://www.rfc-editor.org/rfc/rfc5280.txt. http://www.rfc-editor.org/rfc/rfc5280.txt.

CW05 Chen, J.-C. and Wang, Y.-P., Extensible authentication protocol (eap) and ieee 802.1x: tutorial and empirical experience. *IEEE Communications Magazine*, 43,12(2005), pages supl.26–supl.32.

CW13 Coyne, E. and Weil, T. R., Abac and rbac: scalable, flexible, and auditable access management. *IT Professional*, 15,3(2013), pages 0014–16.

Don02 Donley, C., *Ldap Progamming, Management and Integration*. Manning Publications Co., 2002.

DR08 Dierks, T. and Rescorla, E., The transport layer security (tls) protocol version 1.2. RFC 5246, RFC Editor, August 2008. URL http://www.rfc-editor.org/rfc/rfc5246.txt. http://www.rfc-editor.org/rfc/rfc5246.txt.

EKA14 Ekberg, J.-E., Kostiainen, K. and Asokan, N., The untapped potential of trusted execution environments on mobile devices. *IEEE Security & Privacy*, 12,4(2014), pages 29–37.

Est10 Estrin, D. L., Participatory sensing: Applications and architecture. *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, New York, NY, USA, 2010, ACM, pages 3–4, URL http://doi.acm.org/10.1145/1814433.1814435.

FAKS14 Fremantle, P., Aziz, B., KopeckÃœ, J. and Scott, P., Federated identity and access management for the internet of things. *2014 International Workshop on Secure Internet of Things*, Sept 2014, pages 10–17.

FK11 Frankel, S. and Krishnan, S., Ip security (ipsec) and internet key exchange (ike) document roadmap. RFC 6071, RFC Editor, February 2011. URL http://www.rfc-editor.org/rfc/rfc6071.txt. http://www.rfc-editor.org/rfc/rfc6071.txt.

GHJV95 Gamma, E., Helm, R., Johnson, R. and Vlissides, J., *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

GPRA98 Ghormley, D. P., Petrou, D., Rodrigues, S. H. and Anderson, T. E., Slic: An extensibility system for commodity operating systems. *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '98, Berkeley, CA, USA, 1998, USENIX Association, pages 4–4, URL http://dl.acm.org/citation.cfm?id=1268256.1268260.

GTM11 Guinard, D., Trifa, V., Mattern, F. and Wilde, E., From the internet of things to the web of things: Resource-oriented architecture and best practices. In *Architecting the Internet of Things*, Uckelmann, D., Harrison, M. and Michahelles, F., editors, Springer Berlin Heidelberg, 2011, pages 97–129, URL http://dx.doi.org/10.1007/978-3-642-19157-2_5.

Har12 Hardt, D., The OAuth 2.0 Authorization Framework. RFC 6749, RFC Editor, October 2012. URL https://tools.ietf.org/html/rfc6749.html.

RESTX Hal Lockhart, Bill Parducci, R. S., Rest profile of xacml v3.0 version 1.0. edited by Rémon sinnema. 23 november 2014. oasis committee specification 02., November 2014. http://docs.oasis-open.org/xacml/xacml-rest/v1.0/cs02/xacml-rest-v1.0-cs02.html. Latest version: http://docs.oasis-open.org/xacml/xacml-rest/v1.0/xacml-rest-v1.0.html

HMMC16 Hardjono, T., Maler, E., Machulak, M. and Catalano, D., User-managed access (uma) profile of oauth 2.0. Internet-Draft draft-hardjono-oauth-umacore-14, IETF Secretariat, January 2016. URL http://www.ietf.org/internet-drafts/draft-hardjono-oauth-umacore-14.txt. http://www.ietf.org/internet-drafts/draft-hardjono-oauth-umacore-14.txt.

HSH+08 Hwang, J. Y., Suh, S. B., Heo, S. K., Park, C. J., Ryu, J. M., Park, S. Y. and Kim, C. R., Xen on arm: System virtualization using xen hypervisor for arm-based secure mobile phones. *2008 5th IEEE Consumer Communications and Networking Conference*, Jan 2008, pages 257–261.

IMG+07 Iqbal, Z., Mehmood, A., Ghafoor, A., Ahmed, H. F. and Shibli, A., Authenticated service interaction protocol for multi-agent system. *2007 International Symposium on High Capacity Optical Networks and Enabling Technologies*, Nov 2007, pages 1–5.

JBS15 Jones, M., Bradley, J. and Sakimura, N., Json web token (jwt). RFC 7519, RFC Editor, May 2015. URL `http://www.rfc-editor.org/rfc/rfc7519.txt`. `http://www.rfc-editor.org/rfc/rfc7519.txt`.

KBA02 Kiriansky, V., Bruening, D. and Amarasinghe, S. P., Secure execution via program shepherding. *Proceedings of the 11th USENIX Security Symposium*, Berkeley, CA, USA, 2002, USENIX Association, pages 191–206, URL `http://dl.acm.org/citation.cfm?id=647253.720293`.

KCW10 Kuhn, D. R., Coyne, E. J. and Weil, T. R., Adding attributes to role-based access control. *Computer*, 43,6(2010), pages 79–81.

KMRT15 Kamal, P., Mustafiz, S., Rahman, F. M. A. and Taher, R., Evaluating the efficiency and effectiveness of a federated sso environment using shibboleth. *Journal of Information Security*, 6,3(2015), page 166.

KR15 Karen Rose, Scott Eldridge, L. C., THE INTERNET OF THINGS:AN OVERVIEW understanding the issues and challenges of a more connected world, 2015. URL `http://www.internetsociety.org/sites/default/files/ISOC-IoT-Overview-20151022.pdf`.

Kri17 Kristopher, S., Why oauth 2.0 is vital to iot security, March 2017. URL `http://nordicapis.com/why-oauth-2-0-is-vital-to-iot-security/`.

KVS16 Kumar, S. A., Vealey, T. and Srivastava, H., Security in internet of things: Challenges, solutions and future directions. *2016 49th Hawaii International Conference on System Sciences (HICSS)*, Jan 2016, pages 5772–5781.

AIM10 Luigi, A., Antonio, I. and Giacomo, M., The internet of things: a survey. *Elsevier International Journal of Computer and Telecommunications Networking*, 54,15(2010), pages 2787–2805.

LDJ14 Luhach, A. K., Dwivedi, S. K. and Jha, C., Designing a logical security framework for e-commerce system based on soa. *International Journal on Soft Computing*, 5,2(2014), page 1.

LST16 L. Seitz, G. Selander, E. W. S. E. and Tschofenig, H., Authentication and Authorization for Constrained Environments (ACE). RFC, ACE Working Group , October 2016. URL `https://tools.ietf.org/html/draft-ietf-ace-oauth-authz-04`.

mal08 Extensible authentication protocol overview, October 2008. https://technet.microsoft.com/en-us/library/bb457039.aspx.

Meo16 Meola, A., The $6 trillion opportunity in the IoT. *BI Intelligence, Business Insider*. URL `http://www.businessinsider.com/iot-ecosystem-what-is-the-internet-of-things-2016-2?IR=T\&gt`.

MM06 McCamant, S. and Morrisett, G., Evaluating sfi for a cisc architecture. *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006, USENIX Association, URL `http://dl.acm.org/citation.cfm?id=1267336.1267351`.

MMST16 Mineraud, J., Mazhelis, O., Su, X. and Tarkoma, S., A gap analysis of internet-of-things platforms. *Comput. Commun.*, 89,C(2016), pages 5–16. URL `http://dx.doi.org/10.1016/j.comcom.2016.03.015`.

MSCS16 Maass, M., Sales, A., Chung, B. and Sunshine, J., A systematic analysis of the science of sandboxing. *PeerJ Computer Science*, 2, page e43.

MT15 Mineraud, J. and Tarkoma, S., Toward interoperability for the internet of things with meta-hubs. *CoRR*, abs/1511.08063. URL `http://arxiv.org/abs/1511.08063`.

MWC13 Ma, M., Wang, P. and Chu, C. H., Data management for internet of things: Challenges, approaches and opportunities. *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, Aug 2013, pages 1144–1151.

MZ06 Melnikov, A. and Zeilenga, K., Simple authentication and security layer (sasl). RFC 4422, RFC Editor, June 2006.

Nai15 Nair, S., Using json and rest profiles for external authorization, 2015. https://www.axiomatics.com/blog/entry/using-json-and-rest-profiles-for-external-authorization.html. [14.1.2017]

NT94 Neuman, B. C. and Ts'o, T., Kerberos: An authentication service for computer networks. *IEEE Communications magazine*, 32,9(1994), pages 33–38.

NYHR05 Neuman, C., Yu, T., Hartman, S. and Raeburn, K., The kerberos network authentication service (v5). RFC 4120, RFC Editor, July 2005. URL http://www.rfc-editor.org/rfc/rfc4120.txt. http://www.rfc-editor.org/rfc/rfc4120.txt.

OMEO17 Ouaddah, A., Mousannif, H., Elkalam, A. A. and Ouahman, A. A., Access control in the internet of things: Big challenges and new opportunities. *Computer Networks*, 112, pages 237 – 262. URL http://www.sciencedirect.com/science/article/pii/S1389128616303735.

Pap12 Papazoglou, M., *Web services and SOA : principles and technology*. Pearson Education, Essex, England New York, 2012.

PHG16 Patil, P., Hakiri, A. and Gokhale, A., Cyber foraging and offloading framework for internet of things. *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, June 2016, pages 359–368.

XACML Rissanen, E., extensible access control markup language (xacml) version 3.0. 22 january 2013. oasis standard., January 2013. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

San05 Sankar, K. *Cisco wireless LAN security*, chapter EAP Authentication Protocols for WLANs, pages 157–192. Cisco Press, 2005.

SBJ+14 Sakimura, N., Bradley, J., Jones, M., de Medeiros, B. and Mortimore, C., Openid connect core 1.0. *The OpenID Foundation*, page S3.

SBM+04 Singh, I., Brydon, S., Murray, G., Ramachandran, V., Violleau, T. and Stearns, B., *Designing Web Services with the J2EE 1.4 Platform: JAX-RPC, XML Services, and Clients*. Pearson Education, 2004.

Ser06 Sermersheim, J., Lightweight directory access protocol (ldap): The protocol. RFC 4511, RFC Editor, June 2006. URL `http://www.rfc-editor.org/rfc/rfc4511.txt`. `http://www.rfc-editor.org/rfc/rfc4511.txt`.

Sin15 Sinitsyn, V., Jailhouse. *Linux J.*, 2015,252(2015). URL `http://dl.acm.org/citation.cfm?id=2775334.2775336`.

SN05 Smith, J. E. and Nair, R., The architecture of virtual machines. *Computer*, 38,5(2005), pages 32–38.

SPB+16 Singh, J., Pasquier, T., Bacon, J., Ko, H. and Eyers, D., Twenty security considerations for cloud-supported internet of things. *IEEE Internet of Things Journal*, 3,3(2016), pages 269–284.

SPF+07 Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A. and Peterson, L., Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. *SIGOPS Oper. Syst. Rev.*, 41,3(2007), pages 275–287. URL `http://doi.acm.org/10.1145/1272998.1273025`.

SRGCP15 Sicari, S., Rizzardi, A., Grieco, L. and Coen-Porisini, A., Security, privacy and trust in internet of things: The road ahead. *Computer Networks*, 76, pages 146 – 164. URL `http://www.sciencedirect.com/science/article/pii/S1389128614003971`.

SS13 Seugwon Shin1, Phillip Porras, V. Y. M. F. G. G. M. T., Fresco: Modular composable security services for software-defined networks. *Proceedings of the 20th Annual Network & Distributed System Security Symposium*, San Diego, CA, USA, 2013, Internet Society.

SSQA16 Saadeh, M., Sleit, A., Qatawneh, M. and Almobaideen, W., Authentication techniques for the internet of things: A survey. *2016 Cybersecurity and Cyberforensics Conference (CCC)*, Aug 2016, pages 28–34.

SWGI+16 Shu, R., Wang, P., Gorski III, S. A., Andow, B., Nadkarni, A., Deshotels, L., Gionta, J., Enck, W. and Gu, X., A study of security isolation

techniques. *ACM Comput. Surv.*, 49,3(2016), pages 50:1–50:37. URL http://doi.acm.org/10.1145/2988545.

TEG+06 Tuttle, S., Ehlenberger, A., Gorthi, R., Leiserson, J., Macbeth, R., Owen, N., Ranahandola, S., Storrs, M., Yang, C. et al., *Understanding LDAP-design and implementation*. IBM Redbooks, 2006.

TM15 Tschofenig, H., A. J. T. D. and McPherson, D., Architectural Considerations in Smart Object Networking. RFC 7452, RFC Editor, March 2015. URL http://www.rfc-editor.org/info/rfc7452.

TWS16 Toumassian, S., Werner, R. and Sikora, A., Performance measurements for hypervisors on embedded arm processors. *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Sept 2016, pages 851–855.

VBM15 Vokorokos, L., Baláž, A. and Madoš, B., Application security through sandbox virtualization. *Acta Polytechnica Hungarica*, 12,1(2015), pages 83–101.

VN09 Viswanathan, A. and Neuman, B., A survey of isolation techniques. *Information Sciences Institute, University of Southern California*.

Web10 Weber, R. H., Internet of things — new security and privacy challenges. *Computer Law & Security Review*, 26,1(2010), pages 23 – 30. URL http://www.sciencedirect.com/science/article/pii/S0267364909001939.

WLAG93 Wahbe, R., Lucco, S., Anderson, T. E. and Graham, S. L., Efficient software-based fault isolation. *SIGOPS Oper. Syst. Rev.*, 27,5(1993), pages 203–216. URL http://doi.acm.org/10.1145/173668.168635.

ZCB10 Zhang, Q., Cheng, L. and Boutaba, R., Cloud computing: state-of-the-art and research challenges. *Springer Journal of Internet Services and Applications*, 1,1(2010), pages 7–18. URL http://dx.doi.org/10.1007/s13174-010-0007-6.